

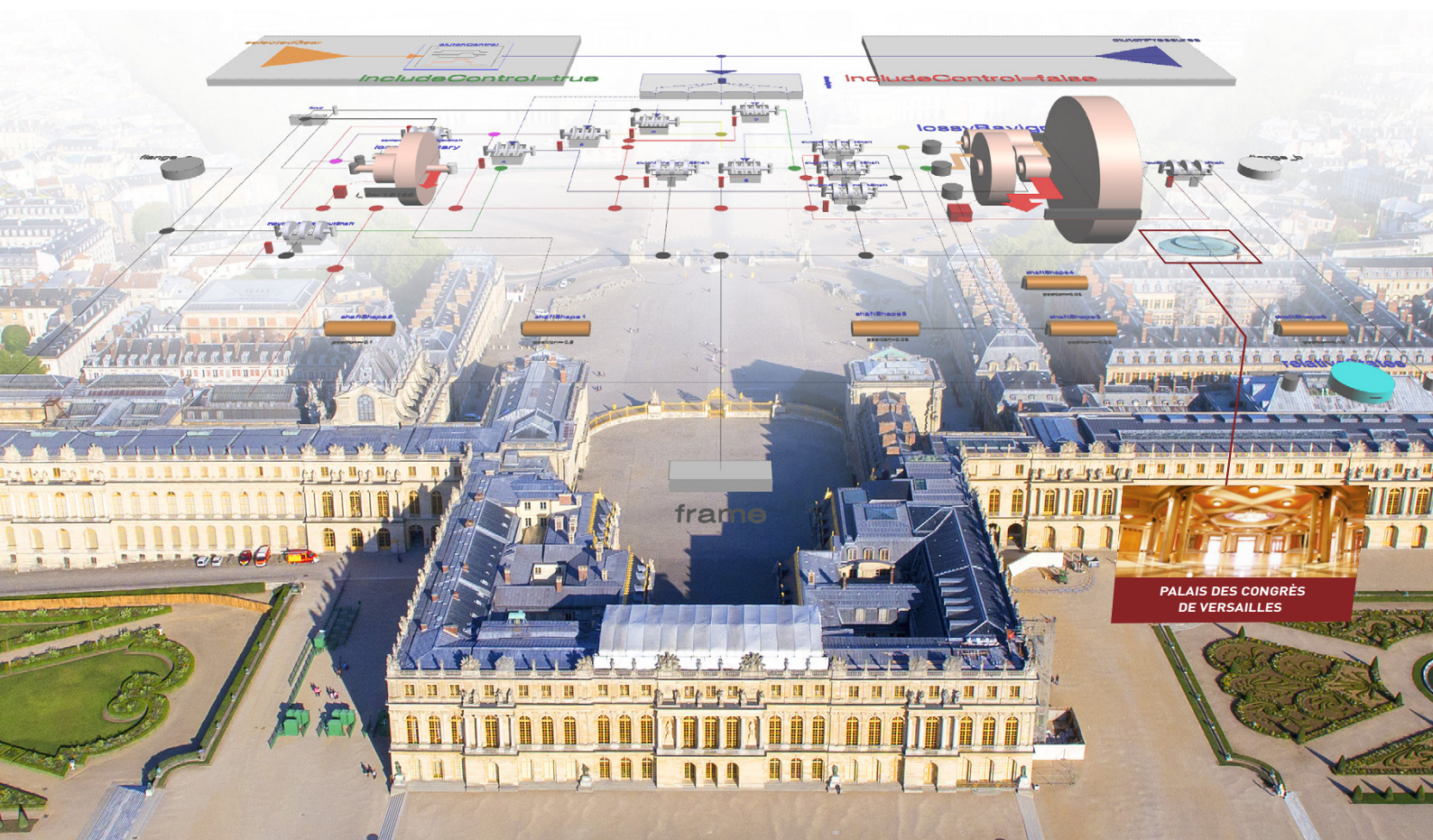
PROCEEDINGS OF THE

11th INTERNATIONAL MODELICA CONFERENCE

September 21–23, 2015

Palais des Congrès de Versailles, France

www.modelica.org



EDITORS: PROF. PETER FRITZSON AND DR. HILDING ELMQVIST



The conference is organized by Dassault Systèmes and Linköping University in cooperation with the Modelica Association.

Proceedings of the 11th International Modelica Conference
Versailles, France, September 21-23, 2015

Editors:

Prof. Peter Fritzson and Dr. Hilding Elmqvist

Published by:

Modelica Association and Linköping University Electronic Press

ISBN: 978-91-7685-955-1

Series: Linköping Electronic Conference Proceedings, No 118

ISSN: 1650-3686

eISSN: 1650-3740

DOI: <http://dx.doi.org/10.3384/ecp15118>

Organized by:

Dassault Systèmes

and

Linköping University

10 rue Marcel Dassault, CS 40501
78946 Vélizy-Villacoublay Cedex
France

Programming Environments Laboratory (PELAB)
Department of Computer and Information Science
SE-581 83 Linköping
Sweden

in co-operation with:

Modelica Association

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

Conference location:

Palais des Congrès de Versailles
10 Rue de la Chancellerie
78000 Versailles
France

Copyright © Modelica Association, 2015

WELCOME

The 11th International Modelica Conference, which takes place at Palais des Congrès de Versailles, is the main event for the Modelica community. Users, library developers, tool vendors, and language designers gather to share their knowledge and learn about the latest scientific and industrial progress related to Modelica and FMI (Functional Mockup Interface).

The fundamental idea behind Modelica is to allow storing modeling knowhow in a high-level formally defined format, i.e., to collect information which you otherwise would find in engineering books only accessible by humans. By allowing convenient reuse of this knowhow by definition of component model libraries, enormous saving in man-hours for setting up simulation studies is achieved. Furthermore, by proper validation of such model libraries, much more reliable conclusions can be made from simulation studies leading to better products and user experience. These considerations lead to the equation-based object-oriented formalism of Modelica.

Since the start of the collaborative design work for Modelica in 1996, 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. In addition, the standardization of the language by the non-profit organization Modelica Association enables Modelica models to be portable between a growing number of tools. Modelica is now used in many industries including automotive, energy and process, aerospace, and industrial equipment. Modelica is the language of choice for model-based systems engineering.

The FMI standard has been added to the project portfolio of the Modelica Association. FMI provides a complementary standard that enables deployment of pre-compiled high quality models originating from different model formats to a larger number of engineers working with system design and verification.

The format of the conference is somewhat changed compared to previous years. We moved the vendor sessions to the first day of the conference to have two days of purely scientific presentations. Starting the tutorials one hour earlier allowed us to allocate more time and have room for 15 vendors to present their offers compared to 6 at the previous conference.

The program is available in an event app for smartphones, tablets, and PCs. It enables searching for papers with abstracts, authors, and conference rooms. It also allows setting up your own schedule by selecting your favorite presentations.

Taking a walk in the Garden of Versailles is suggested on Tuesday evening. We have allocated a break of more than one hour after the scientific program before the conference dinner is served at the Palais des Congrès de Versailles. This means that you have time to see the Apollo Fountain.

Conference highlights:

- 2 Keynote speeches
- 83 papers in 4 parallel tracks
- 18 posters
- 7 tutorials
- 5 libraries submitted for the Modelica Library Award
- 15 vendor sessions presenting the latest Modelica and FMI tools
- A fully booked exhibition area featuring 20 exhibitors
- Electronic proceedings including all papers and some associated Modelica libraries and models

Finally, we want to acknowledge the support we received from the conference board and program committee. Special thanks to our colleagues at this year's organizers, Dassault Systèmes and Linköping University, and Amelie Rönngård from Altitude Meetings. The support from the conference sponsors is gratefully acknowledged. Last but not least, thanks to all authors, keynote speakers, and presenters for their contributions to this conference.

We wish all participants an enjoyable and inspiring conference.

Lund and Linköping, September 1, 2015
Hilding Elmqvist and Peter Fritzson



Hilding Elmqvist



Peter Fritzson

KEYNOTE SPEAKERS



Designing Cyber-Physical Systems: A Tale of Two Worlds Coming Together

Presenter:

Prof. Alberto Sangiovanni-Vincentelli
UC Berkeley, USA

Abstract: Cyber-Physical Systems have been the focus of many research and public forum initiatives in the world since the early 2000s. The concept of CPS involves the tight integration and co-design of physical (for example, mechanical, electrical, biological and chemical), systems with analysis, monitoring and control implemented on a computing system. As such it has important intersections with other fields of great interests such as Internet of Things, Hybrid Systems, Swarm Systems and Systems of Systems. One of the main challenges has been to develop solid foundations for design and manufacturing including formal methods and requirement capture.

I will review the major directions of research and industrial relevance of CPS with particular attention to design methodologies and requirement capture with considerations about approaches to CPS simulation and analysis and their limitations.

Bio: Alberto Sangiovanni-Vincentelli holds the Buttner Chair of EECS, University of California, Berkeley. He was a co-founder of Cadence and Synopsys, the two leading companies in Electronic Design Automation. He was a member of the HP Strategic Technology Advisory Board, of the Science and Technology Advisory Board of GM, and is a member of the Technology Advisory Council of UTC. He is member of the Scientific Council of the Italian National Science Foundation (CNR) and of the Executive Committee of the Italian Institute of Technology. He is President of the Consiglio Nazionale Garanti della Ricerca, and of the Strategic Committee of the Italian Strategic Fund.

He received the Kaufman Award for "pioneering contributions to EDA", the IEEE/RSE Maxwell Medal" for groundbreaking contributions that have had an exceptional impact on the development of electronics and electrical engineering or related fields. He holds an honorary Doctorate by the University of Aalborg, Denmark and one by KTH, Sweden.

He is an author of over 850 papers, 17 books and 2 patents, is IEEE Fellow and a Member of the NAE.



A systems engineering perspective for Modelica and the heritage of synchronous language

Presenter:

Dr. Albert Benveniste,
INRIA, France

Abstract: In the first part of my talk I shall develop a vision of the central role of Modelica in systems engineering. The Modrio project has recently developed a Requirements profile for Modelica and progresses have recently been made regarding the link between Modelica and safety analyses. I shall discuss how far, I think, one could go in these directions. I shall also draw directions toward using Modelica for system-wide monitoring and diagnosis. All of this calls for a rigorous understanding of Modelica, its execution semantics: paying attention to this will constitute the second part of my presentation. I shall describe the background from synchronous languages by emphasizing how sound compilation schemes can be formally derived and how some of the above mentioned uses in system engineering were performed with synchronous languages. I shall conclude by indicating how these techniques can be adapted to derive structural analyses for multi-mode DAE systems. Nonstandard analysis will be used to help for this.

Bio: Albert Benveniste was Directeur de Recherche at INRIA, where he is now emeritus. In 1990 he received the CNRS silver medal, he was elected IEEE fellow in 1991 and IFAC Fellow in 2013. From 1986 to 1990 he was vice-chairman of the IFAC committee on Theory and was chairman of this committee for 1991-1993. He has been Associate Editor (at Large) for IEEE Transactions on Automatic Control, Associate Editor for Int. J. of Adaptive Control and Signal Processing, and Int. J. of Discrete Event Dynamical Systems, and member of the Editorial Board of the Proceedings of the IEEE. From 1997 to 2013, he was head for INRIA of the joint Alcatel-INRIA research programme. He is a member of the scientific advisory boards of Safran Group and Orange. From 2011 to 2014, he was co-heading the Center of Excellence (Labex) CominLabs in the area of telecommunications and Information systems. He has been elected to the Académie des Technologies in december 2011. His areas of interest cover system identification in control, embedded systems in computer science, and network management in telecommunications.

Program Committee

General Chair

Dr. Hilding Elmqvist, Dassault Systèmes, Lund, Sweden

Program Chair

Prof. Peter Fritzon, Linköping University, Sweden

Program Board

Dr. Hilding Elmqvist, Dassault Systèmes, Lund, Sweden

Prof. Peter Fritzon, Linköping University, Sweden

Prof. Martin Otter, DLR, Germany

Dr. Michael Tiller, Xogeny, Michigan, USA

Program Committee

Prof. Bernhard Bachmann, Univ. Applied Sciences Bielefeld, Bielefeld, Germany

Prof. John Baras, University of Maryland, Maryland, USA

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

Dr. Timothy Bourke, INRIA, France

Daniel Bouskela, EDF R&D, Paris, France

Dr. David Broman, KTH Royal Institute of Technology, Stockholm, Sweden

Dr. Lena Buffoni, Linköping University, Sweden

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. Olaf Enge-Rosenblatt, Fraunhofer IIS Dresden, Dresden, Germany

Prof. Gianni Ferretti, Politecnico di Milano, Italy

Dr. Rüdiger Franke, ABB AG, Mannheim, Germany

Dr. Jens Frenkel, ITI GmbH, Dresden, Germany

Prof. Peter Fritzon, Linköping University, Sweden

Prof. Manfred Hajek, TU Munich, Munich, Germany

Peter Harman, CyDesign, Coventry, United Kingdom

Prof. Anton Haumer, Technical consultant, OTH Regensburg, Regensburg, Germany

Dr. Dan Henriksson, Dassault Systèmes, Lund, Sweden

Dr. Yutaka Hirano, Toyota, Japan

Prof. Bengt Jacobson, Chalmers Technical University, Gothenburg, Sweden

Prof. Tommi Karhela, VTT / Aalto University, Espoo, Finland

Åke Kinnander, Siemens Turbo, Sweden

Dr. Johan de Kleer, PARC, Palo Alto, USA

Dr. Christian Kral, Mechatroniker für Elektromaschinenbau und Automatisierung, Vienna, Austria

Jochen Köhler, ZF AG, Friedrichshafen, 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

Ramine Nikoukhah, Altair Engineering, Paris, France
Prof. Henrik Nilsson, University of Nottingham, Nottingham, Great Britain
Prof. Akira Ohata, Toyota Motor Corporation, Tokyo, Japan
Dr. Hans Olsson, Dassault Systèmes, Lund, Sweden
Prof. Martin Otter, DLR, Germany
Prof. Peter Pepper, TU Berlin, Berlin, Germany
Dr. Andreas Pillekeit, dSPACE, Germany
Dr. Adrian Pop, Linköping University, Sweden
Dr. Adrijan Ribaric, Sentient Science, USA
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
Prof. Stefan-Alexander Schneider, Hochschule Kempten, Kempten, Germany
Dr. Martin Sjölund, Linköping University, Sweden
Prof. Thierry Soriano, Supmeca, France
Dr. Rita Streblov, RWTH Aachen, Aachen, Germany
Dr. Ed Tate, Exa, Livonia, USA
Dr. Wilhelm Tegethoff, TLK-Thermo GmbH, Germany
Bernhard Thiele, Linköping University, Sweden
Dr. Michael Tiller, Xogeny, Michigan, USA
Dr. Jakub Tobolar, DLR, Munich, Germany
Dr. Hubertus Tummescheit, Modelon Inc., West Hartford, USA
Prof. Alfonso Urquía, UNED, Spain
Prof. Luigi Vanfretti, KTH- Royal Inst. of Technology, Stockholm, Sweden
Dr. Subbarao Varigonda, Cummins, Columbus, USA
Dr. Stéphane Velut, Lund, Sweden
Dr. Michael Wetter, LBNL, Berkeley, USA
Dr. Dirk Zimmer, DLR, Germany
Dr. Johan Åkesson, Modelon AB, Lund, Sweden

Conference Organization Team:

Amelie Rönngard, Malmö, Sweden
MCI France
Dr. Hilding Elmqvist, Dassault Systèmes, Lund, Sweden
Prof. Peter Fritzson, Linköping University, Sweden
Martin Malmheden, Dassault Systèmes, Paris, France
Dr. Martin Sjölund, Linköping University, Sweden
Bernhard Thiele, Linköping University, Sweden
Ulrika Wiklund, Dassault Systèmes, Lund, Sweden

Contents

Session 2A: FMI 1	17
Experience with Industrial In-House Application of FMI	17
A Novel Proposal on how to Parameterize Models in Dymola Utilizing External Files under Consideration of a Subsequent Model Export using the Functional Mock-Up Interface	23
Design Choices for Thermofluid Flow Components and Systems that are Exported as Functional Mockup Units	31
FMI for Physical Models on Automotive Embedded Targets	43
Session 2B: Building Energy Applications 1	51
Methodology for Obtaining Linear State Space Building Energy Simulation Models	51
Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation	59
Energy Efficient Design for Hotels in the Tropical Climate using Modelica	71
Presentation, Validation and Application of the DistrictHeating Modelica Library	79
Session 2C: Simulation Techniques	89
Multi-Mode DAE Systems with Varying Index	89
Internalized State-Selection: Generation and Integration of Quasi-Linear Differential-Algebraic Equations	99
Fractional-Order Modelling in Modelica	109
Modelica Library for Feed Drive Systems	117
Session 2D: Automotive Applications 1	127
Model-based Development of a Holistic Thermal Management System for an Electric Car with a High Temperature Fuel Cell Range Extender	127
Predicting the Effect of Gearbox Preconditioning on Vehicle Efficiency	135
Model Based Development of Future Small Electric Vehicle by Modelica	143
Modelling of Torque-Vectoring Drives for Electric Vehicles: a Case Study	151
Session 3A: FMI 2	159
Co-Simulation of Hybrid Systems with SpaceEx and Uppaal	159
Automated Deployment of Modelica Models in Excel via Functional Mockup Interface and Integration with modeFRONTIER	171
An Open-Source Graphical Composite Modeling Editor and Simulation Tool Based on FMI and TLM Co-Simulation	181
The Modelica Language and the FMI Standard for Modeling and Simulation of Smart Grids	189
Session 3B: Building Energy Applications 2	197
Coupled modeling of a District Heating System with Aquifer Thermal Energy Storage and Absorption Heat Transformer	197
Energy-Efficient Design of a Research Greenhouse with Modelica	207
Production Planning for Distributed District Heating Networks with JModelica.org	217
Hardware-in-the-Loop-Simulation of a Building Energy and Control System to Investigate Circulating Pump Control Using Modelica	225
Session 3C: Modelica Language & Compiler Implementation 1	235
Automatic GPU Code Generation of Modelica Functions	235
Constructs for Meta Properties Modeling in Modelica	245
Flattening of Modelica State Machines: A Practical Symbolic Representation	255
Exploiting Repeated Structures and Vectorization in Modelica	265
Session 3D: Automotive Applications 2	273
High Fidelity Multibody Vehicle Dynamics Models for Driver-in-the-Loop Simulators	273
Modeling and Validation of a Multiple Evaporator Refrigeration Cycle for Electric Vehicles	281
Modeling the Effects of Energy Efficient Glazing on Cabin Thermal Energy & Vehicle Efficiency	291

Session 4A: Optimization Applications and Methods	301
A Framework for Nonlinear Model Predictive Control in JModelica.org	301
A Toolchain for Solving Dynamic Optimization Problems Using Symbolic and Parallel Computing . .	311
NMPC Application using JModelica.org: Features and Performance	321
Session 4B: Control Applications 1	329
A Modelica Library for Manual Tracking	329
Model-based control with FMI and a C++ runtime for Modelica	339
Nonlinear Dynamic Inversion Control for Wind Turbine Load Mitigation based on Wind Speed Mea- surement	349
Session 4C: Novel Modelica Applications and Libraries	359
Free Modelica Library for Chemical and Electrochemical Processes	359
Modeling Biology in Modelica: The Human Baroreflex	367
A City Traffic Library	377
Session 4D: Building Energy Applications 3	383
An Open Toolchain for Generating Modelica Code from Building Information Models	383
Lessons Learnt from Network Modelling of a Low Heat Density District Heating System	393
Modelica based Design and Optimisation of Control Systems for Solar Heat Systems and Low Energy Buildings	401
Session 5A: Control Applications 2	411
How to Shape Noise Spectra for Continuous System Simulation	411
Dynamic Modelling of a Flat-Plate Solar Collector for Control Purposes	419
Session 5B: Mechanical Systems	427
Generic Modelica Framework for MultiBody Contacts and Discrete Element Method	427
Different Models of a Scaled Experimental Running Gear for the DLR RailwayDynamics Library . . .	441
Session 5C: Modelica Language & Compiler Implementation 2	449
Efficient Compilation of Large Scale Dynamical Systems	449
Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives	459
Session 5D: Electrical Systems	469
Developing Mathematical Models of Batteries in Modelica for Energy Storage Applications	469
Average Model of a Synchronous Half-Bridge DC/DC Converter Considering Losses and Dynamics . .	479
Session 7A: Aerospace Applications 1	485
Modeling and Simulation of Liquid Propellant Rocket Engine Transient Performance Using Modelica .	485
Model Based Specifications in Aircraft Systems Design	491
Session 7B: Electrical Machines	501
Multi Electrical Machine Pre-Design Tool with Error Handling and Machine Specific Advanced Graph- ical Design Aid Features Based on Modelica	501
Enhancements of Electric Machine Models: The EMachines Library	509
Session 7C: 3D Representations for Modelica Models	517
Simulation of Piping 3D Designs Powered by Modelica	517
3D Schematics of Modelica Models and Gamification	527
Session 7D: Virtual Test Benches	537
Holistic Virtual Testing and Analysis of a Concept Hybrid Electric Vehicle Model	537
Modeling of an Automatic Transmission for the Evaluation of Test Procedures in a Virtual End-of-Line Test Bench	547
Session 8A: Aerospace Applications 2	557
A New Fault Injection Method for Liquid Rocket Pressurization and Feed System	557
Automated Safety Analysis by Minimal Path Set Detection for Multi-Domain Object-Oriented Models	565
High-fidelity Modelling of Self-regulating Pneumatic Valves	577

Session 8B: Power, Energy & Process Applications 1	585
Dynamic Modeling of a Central Receiver CSP System in Modelica	585
Modeling of Linear Concentrating Solar Power using Direct Steam Generation with Parabolic-Trough	595
Transient Simulation of the Power Block in a Parabolic Trough Power Plant	605
Session 8C: Safety & Formal Methods	615
Fault Detection and Diagnosis with Modelica Language using Deep Belief Network	615
Formal Requirements Modeling for Simulation-Based Verification	625
Towards a Formalized Modelica Subset	637
Session 8D: Thermofluid Systems, Models and Libraries 1	647
Fundamental EoS Implementation for {Water+Ammonia} in Modelica	647
MultiComponentMultiPhase - A Framework for Thermodynamic Properties in Modelica	653
Modeling of the German National Standard for High Pressure Natural Gas Flow Metering in Modelica	663
Session 10A: Testing & Diagnostics	671
Automatic Regression Testing of Simulation Models and Concept for Simulation of Connected FMUs in PySimulator	671
Abrasive Waterjet Intensifier Model for Machine Diagnostics	681
Optimica Testing Toolkit: a Tool-Agnostic Testing Framework for Modelica Models	687
Session 10B: Power, Energy & Process Applications 2	695
Status of the TransiEnt Library: Transient Simulation of Coupled Energy Networks with High Share of Renewable Energy	695
Mathematical Model of Soot Blowing Influences in Dynamic Power Plant Modelling	707
Flexibilization of Coal-fired Power Plants by Dynamic Simulation	715
Session 10C: Modelica Tools	725
Where impact got Going	725
Visualizing Simulation Results from Modelica Fluid Models Using Graph Drawing in Python	737
Reuse of Physical System Models by means of Semantic Knowledge Representation: A Case Study applied to Modelica	747
Session 10D: Thermofluid Systems, Models and Libraries 2	759
Mass Conserving Models of Vapor Compression Cycles	759
EPSILON Modelica Library for Thermal Applications	769
Multi-Objective Optimization of Dynamic Systems combining Genetic Algorithms and Modelica: Ap- plication to Adsorption Air-Conditioning Systems	777
Poster Session	785
A new Modelica Electric and Hybrid Power Trains Library	785
Initiatives for Acausal Model Connection using FMI in JSAE (Society of Automotive Engineers of Japan)	795
Dynamical Model of a Vehicle with Omni Wheels: Improved and Generalized Contact Tracking Algorithm	803
Kansei Modeling for Delight Design based on 1DCAE Concept	811
A Modelica Library Organization Method Supporting Online Modeling and Simulation	817
Control Development and Modeling for Flexible DC Grids in Modelica	823
Towards Enhanced Process and Tools for Aircraft Systems Assessments during very Early Design Phase	831
Using FMI in a Cloud-based Web Application for System Simulation	845
Anticipatory Shifting - Optimization of a Transmission Control Unit for an Automatic Transmission through Advanced Driver Assistance Systems	849
Simulation of Distributed Energy Storage in the Residential Sector and Potential Integration of Gas- based Renewable Energy Technologies using Modelica	855
Test of Basic Co-Simulation Algorithms Using FMI	865
Experimental Calibration of Heat Transfer and Thermal Losses in a Shell-and-Tube Heat Exchanger .	873
Suitability of Different Real-Time Solvers for a Model-Based Engineering Toolchain using Industrial Rexroth Controllers	883
Integrated Engineering based on Modelica	893
Coupling Model Exchange FMUs for Aggregated Simulation by Open Source Tools	903

An Aeronautic Case Study for Requirement Formalization and Automated Model Composition in
Modelica 911
FastHVAC - A Library for Fast Composition and Simulation of Building Energy Systems 921
Open Source Library for the Simulation of Wind Power Plants 929

Author Index

Ackermann, Günter	695	Crespo, Matthieu	831
Ahle, Elmar	43	Croteau, Dominique	189
Åkesson, Johan	687, 903	Cudok, Falk	197
Albarelo, Nicolas	911	Dahlberg, Simon	527
Alvarez Rodríguez, Jose María	747	Dao, Thanh-Son	469
Andersson, Christian	903	Daumas, Julien	831
Andres, Markus	23	de La Calle, Alberto	873
Andresen, Lisa	695	Del Hoyo Arce, Itzal	393, 419
Andrieu, Olivier	637	Delgado Beltran, Juan Gabriel	273
Arumugham, Siva Sankar	43	Dempsey, Mike	135, 273
Asghar, Adeel	181, 671	Dersch, Jürgen	605
Aurousseau, Antoine	595	Diehl, Stephan	23
Axelsson, Magdalena	301	Ding, Ji	485
Bachmann, Bernhard	339	Dolin, Nicolas	769
Baldwin, Alexander D.	527	Dominik, Andreas	367
Barbe, Jean-Baptiste	377	Donn, Christian	537
Bardow, André	777	Dubucq, Pascal	695
Barrios, German	71	Duggan, Alexander	171
Batteh, John	171	Eberhart, Philip	929
Bau, Uwe	777	Edman, Johan	585
Baviere, Roland	79	Elmqvist, Hilding	89, 235, 245, 427, 527, 625
Becker, Leonard	647	Ernst, Gernot	367
Belhaj, Issam	893	Fateh, Nader	171
Bensch, Valerie	537	Feral, Hervé	769
Berg, Lars Fredrik	127	Ferretti, Gianni	681
Bergero, Federico	449	Finkbeiner, Konstantin	921
Bertsch, Christian	43	Fischer, Torben	127
Bezian, Jean-Jacques	595	Fish, Garron	273
Bianconi, Raphael	831	Flehmig, Martin	265
Bittner, Stefan	845	Folie, Michael	537, 849
Blochwitz, Torsten	401	Fontes De Miranda, Pablo	911
Boes, Jérémy	189	Francke, Henning	197
Bogomolov, Sergiy	159	Franke, Rüdiger	339
Bohlin, Markus	217	Friebe, Johannes	537
Bonilla, Javier	873	Fritzson, Dag	181
Bosmans, Maarten	653	Fritzson, Peter	181, 255, 625, 671, 911
Botta, Mariano	449	Fuchs, Marcus	31, 737
Bouskela, Daniel	625	Führer, Claus	903
Braun, Willi	181, 339	Funkquist, Jonas	217
Brecher, Christian	117	Gall, Leo	17
Buffoni, Lena	625, 911	Gallardo-Yances, Stephanie	17
Camilleri, Guy	189	Gallego, Elena	747
Campostrini, Esteban	449	Garcia Espinosa, Antoni	501
Cao, Jun	383	Garro, Alfredo	625
Casella, Francesco	109, 459, 577	Gauterin, Frank	127
Ceraolo, Massimo	785	Geletu, Abebe	311
Chaker, Salim	849	Gengler, Thierry	517
Chapuis, Christophe	517	Gerasimov, Kirill	803
Chen, Liping	485, 557	Ghandriz, Toheed	427
Chilard, Olivier	189	Gierow, Conrad	707
Chung, Tek Shan	929	Giese, Tim	491
Clauß, Christoph	401, 865	Gillot, Romain	135
Colaço, Jean-Louis	637	Giraud, Loic	79
Colleoni, Arnaud	769	Gleizes, Marie-Pierre	189
Constantin, Ana	225	Goesmann, Alexander	367
Corrales Ciganda, José Luis	647	Gohl, Jesse	171

Goletti, Massimo	681	Lanzerath, Franz	777
Göres, Jörn	547	Larsen, Kim G.	159
Görner, Klaus	715	Larsson, Per-Ola	217
Goteman, Axel	235, 427	Laughman, Christopher	759
Götz, Florian	127	Lauster, Moritz	383
Gräber, Manuel	281	Lazutkin, Evgeny	311
Graf, Frank	855	Lee, Byoung Doo	615
Grasso, Marco	681	Lee, Dong Kyu	615
Gravelle, Aled	291	Li, Dan	71
Greitschus, Marius	159	Li, Pu	311
Gubsch, Ines	265	Li, Wan	817
Gühmann, Clemens	547	Liebold, Edgar	401
Hartlep, Christian	321	Link, Kilian	17
Hassel, Egon	707	Liu, Wei	485
Haufe, Jürgen	401	Liyana, Eashan	377
Haumer, Anton	509, 929	Llorens, Juan	747
Heckmann, Andreas	411, 441	López Perez, Susana	393, 419
Helsen, Lieve	51, 59	Magnusson, Fredrik	301
Henningsson, Toivo	301, 321	Maile, Tobias	383
Herfs, Werner	117	Majetta, Kristin	401
Herrero López, Saioa	393, 419	Matejak, Marek	359
Herzog, Hans-Georg	479	Mattsson, Sven Erik	89
Hintzen, Ullrich	401	Menager, Nils	883, 893
Hirano, Yutaka	143, 795	Mengist, Alachew	181, 671
Hirono, Tomohide	795	Mesonero Dávila, Iván	393, 419
Hirsch, Tobias	605	Michalski, Tomasz D.	501
Hofmann, Andreas	893	Mickan, Bodo	663
Höger, Christoph	99	Mikelsons, Lars	883, 893
Hopfgarten, Siegbert	311	Mikučionis, Marius	159
Huang, Sen	71	Miranda, Reymundo	71
Hübel, Moritz	707	Modarrez Razavi, Sara	217
Huber, Frank	849	Möllenbruck, Florian	715
Inderfurth, Alexander	197	Monno, Michele	681
Inoue, Shintaro	143	Moser, Sascha	479
Jardin, Audrey	625	Motschke, Tobias	117
Jensen, Peter G.	159	Mühlbauer, Monika	17
Ježek, Filip	359	Müller, Dirk	225, 383, 737, 921
Johnsson, Anna	823	Murakami, Shitaroh	795
Johnsson, Victor	687	Nagel, Wolfgang E.	265
Jorissen, Filip	51, 59	Neidhold, Thomas	845
Kampfmann, Rüdiger	883	Neitzke, Daniel	777
Kather, Alfons	695	Neudorfer, Jonathan	43
Keck, Alexander	441	Nilsson, Andreas	217
Kehrer, Christian	849	Nocke, Jürgen	707
Klöckner, Andreas	411	Nouidui, Thierry S.	31
Klostermann, Volker	401	Nytsch-Geusen, Christoph	197, 383
Knoblach, Andreas	411	Oeljeklaus, Gerd	715
Koeppel, Wolfgang	855	Oelsner, Olaf	845
Kofman, Ernesto	449	Ohsumi, Yuji	795
Kofranek, Jiri	359	Ohtomi, Koichi	811
Köhler, Jürgen	281	Olenmark, Andreas	823
Kollmeier, Hans-Peter	127	Olsson, Hans	235, 245, 625
Kosenko, Ivan	803	Oppermann, Jens	225
Kral, Christian	509, 929	Ota, Junya	143
Kranz, Stefan	197	Otter, Martin	89, 245, 491, 625
Kuhn, Martin	491	Özdemir, Denis	117
Lachassagne, Laurent	769	Pagano, Bruno	637
Lacroux, Simon	377	Palanisamy, Arunkumar	671

Pannu, Pukashawar	903	Siemers, Alexander	181
Paulus, Cédric	79	Sjölund, Martin	671
Payelleville, Maxime	625	Sloth, Jens	823
Peniche Garcia, Ricardo	695	Spike, Jonathan	537
Perles, Alexandre	189	Starinsk, Andreas	715
Perlman, Alexander	687	Steinbrecher, Andreas	99
Petridis, Kosmas	865	Sten, Jon	687
Pfeiffer, Andreas	671	Stepanov, Sergey	803
Picard, Damien	51	Stinner, Sebastian	921
Picarelli, Alessandro	135, 291	Streblow, Rita	225, 737, 921
Pipiorke, Jörg	207	Strump, Thomas	159
Pitchaikani, Anand	171	Svensson, Jörgen	823
Polklas, Thomas	605	Taube, Julian	479
Pollok, Alexander	109, 235, 577	Tavella, Jean-Philippe	189
Pop, Adrian	181, 255, 671	Teraoka, Yoichi	795
Potter, James	329	Thiele, Bernhard	255
Prabhakaran, Praseeth	855	Thomas, Eric	625, 831
Prölss, Katrin	653	Thomas, Olivier	831
Qiao, Hongtao	759	Thorade, Matthias	383
Rachkov, Alexey	803	Thuy, Andreas	43
Rädler, Jörg	383	Thuy, Nguyen	625
Ramachandran, Karthikeyan	43	Tifan, Xiong	817
Reiner, Matthias	349	Tiller, Michael	725
Remmen, Peter	383	Tilly, Anders	687
Remond, Xavier	517	Tobolář, Jakub	151
Riba Ruiz, Jordi-Roger	501	Tribula, Martin	359
Ribas Tugores, Carles	197	Tripakis, Stavros	159
Richter, Marcel	715	Tummescheit, Hubertus	653
Robinson, Dr. Simon	291	Tundis, Andrea	625
Roca, Lidia	873	Unger, Rene	207
Rodríguez-García, Margarita M.	873	Valenzuela, Loreto	873
Romeral Martinez, Luis	501	Vallée, Mathieu	79
Röper, Jan	547	van der Linden, Franciscus L. J.	151
Roxling, Vilhelm	235, 427	van Es, Eli	653
Runvik, Håkan	217	van Treeck, Christoph	383
Satabin, Lucas	637	Varchmin, Andreas	281
Scaglioni, Bruno	681	Velut, Stéphane	217
Schallert, Christian	565	von der Heyde, Michael	663
Schamai, Wladimir	625, 911	Vuillerme, Valéry	595
Schenk, Heiko	605	Walther, Marcus	265, 339
Schmitke, Chad	469, 537	Waurich, Volker	265
Schmitt, Thomas	23	Wetter, Michael	31, 59
Schmitz, Gerhard	663, 695	Wilhelmsson, Carl	823
Schneider, Georg Ferdinand	225	Wimmer, Reinhard	383
Schoenewolf, Stefan	479	Windahl, Johan	585, 653
Schölzel, Christopher	367	Winkler, Dietmar	725
Schuchart, Joseph	265	Winter, Michael	479
Schumacher, Markus	921	Worschech, Niklas	339, 883
Schwan, Torsten	207	Xie, Gang	485, 557
Schwarz, Christine	537	Yang, Hao	485
Schwarz, Christoph	441	Yongchao, Li	817
Seidel, Stephan	401	Zhang, Haiming	485
Sekisue, Takayuki	795	Zhiming, Zhou	817
Sewgobind, Awin	653	Zhou, Fanli	557
Seya, Osamu	795	Zhu, Mingqing	557
Shao, Jintao	557	Ziegler, Stephan	23
Shimada, Satoshi	795	Zimmer, Dirk	109, 235, 349
Shin, Jin Woo	615	Zuo, Wangda	71

Experience with Industrial In-House Application of FMI

Kilian Link¹ Leo Gall² Monika Mühlbauer³ Stephanie Gallardo-Yances⁴

^{1,3,4}Siemens AG, Germany, {kilian.link, monika.muehlbauer, stephanie.gallardo}@siemens.com

²LTX Simulation GmbH, Germany, leo.gall@ltx.de

Abstract

This paper discusses FMI usage in an in-house simulation tool landscape where it helps to open doors between different tools. However, limitations due to missing physical connectors or missing model structure are faced and are described with the help of use cases.

Information hiding in FMI can turn out obstructive in in-house applications. Experiences from implementing FMI support in in-house simulation tools are shared.

Keywords: FMI in in-house simulators, FMI2.0, physical connectors, structured parameters, co-simulation, model exchange, NMPC, control test

1 Introduction

The industrial application of Functional Mock-Up Interface (FMI) has already been discussed several times, e.g. (Bertsch et al., 2014). The goal of this paper is to add another aspect to this discussion. Our focus lies on the in-house applicability of FMI in coupling different tools and propriety models.

For several years, we are developing side-by-side two different simulation tools for power plant systems. An in-house Modelica library, called SiemensPower, and a C++ based in-house tool for a similar purpose named Dynaplant. The reason for developing both simulation tools lies in their different strengths and weaknesses. Dynaplant scores with its high performance and numerical robustness that allows us the investigation of fluid systems with a hundred thousand dynamic states or more. It is fully integrated into the in-house tool chain and allows an automatic model setup from the design software KRAWAL[®] for steady-state heat balances. The GUI is optimized for the purpose of modeling large fluid systems and the plant model looks very similar to the familiar KRAWAL[®] model. The drawback of Dynaplant is the relatively high effort when developing new component models.

The main driver for starting a Modelica library was the modeling flexibility and transparency to the user. Naturally, a Modelica library requires less effort in, developing and maintaining of models. However, the performance and numerical robustness is worse.

Once the need for two simulation environments is accepted, it stands to reason to combine them in order to leverage the benefits of both solutions. The first step

is to integrate Modelica models into Dynaplant to overcome its limits with respect to modeling flexibility (Sun et al., 2011). This development was started some years ago based on Dymola's code export feature. Then, FMI for Model Exchange was adopted as soon it became available. The use cases in chapter 2.1 explain this application in more detail.

Chapter 2.2 addresses the second class of use cases, which also deal with tool interoperability. However, in these cases co-simulation Functional Mock-up Units (FMU) are exported from Dynaplant models and serve as a representation of the real plant in the loop with a Nonlinear Model Predictive Controller (NMPC).

Chapter 3 focuses on the development of FMI support in Dynaplant and points out some issues experienced during implementation.

2 Use Cases

Two kinds of use cases are described in this chapter, one targeting the utilization of FMI for model exchange, the other addressing FMI for co-simulation, see Figure 1. Apart from different application areas major similarities exist:

- The use cases focus on tool interoperability
- All FMUs reside in-house only
- The models are huge with respect to number of parameters, dynamic states and internal/local variables

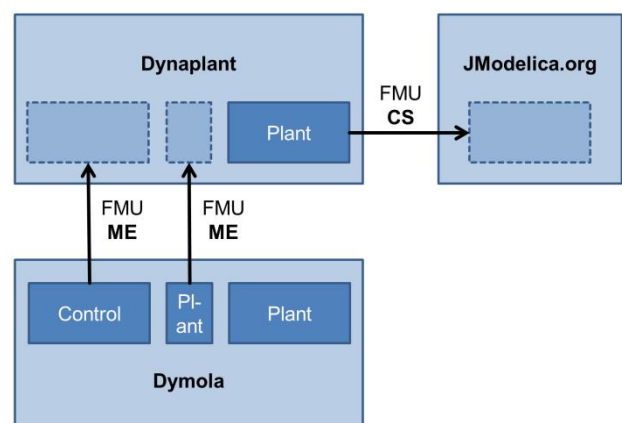


Figure 1: Overview of Use Cases

Our requirements of model exchange and tool exchange certainly differ in some aspects from those in

the use cases defined by the Modelisar project which are described in (Blochwitz et al., 2011). For our use cases there is no necessity for information hiding or IP protection. On the contrary, the goal is to keep as much information available for the user as possible.

2.1 Dynaplant Simulation

Our in-house simulator Dynaplant has been developed for more than ten years in a cooperation of Siemens Power and Gas and Siemens Corporate Technology. In the past, the analysis focused on the detailed, dynamic behavior of the water-steam cycle in a combined-cycle power plant.

A special emphasis has been drawn on the investigation of hydro-thermal dynamic-stability of once-through evaporators between gas and water side (Franke, 2008). Lately, the scope was extended to investigations on the plant level. In general, this leads to a higher need for flexibility in modeling because different subsystems come into focus depending on the application. FMI is an enabling technique to provide that flexibility, e.g. to add the gas side and also advanced controls, whilst keeping the efforts low. The simulation support of tight project schedules was only possible through the usage of FMUs.

In Dynaplant, components are modeled in one-dimensional resolution in an acausal way. The GUI is written in C# using a Microsoft Visio Add-in whereas the plant model itself is stored in a Modelica similar format. For simulation the plant is translated to C++.

2.1.1 Physical Component Models

The most natural use case for FMI in in-house simulators is to import models from different sources via model exchange. By this means, we integrate Modelica models of subsystems in Dynaplant to extend its application scope. The obvious driver behind this is the wish to benefit from both, the high performance of the C++ based in-house simulator and the modeling flexibility of Modelica.

Figure 2 shows the Modelica model of a multistage pump system controlled by a variable gear modelled in Dymola. The model has been prepared for FMI export which required an expansion of the physical fluid ports to a signal interface, already resolving the causality. Even for the very simple interface of such a pump system the transformation to a pure signal interface adds some overhead. Additionally, it becomes harder to understand the model when importing it as an FMU, see Figure 3. In Figure 4 the same system is modelled based on built-in components. The limitation of FMUs to signal interfaces very much hinder a convenient use as an imported component. These issues will boost the demand to provide all needed models as built-in components and not as FMUs. All the more, since the

pump example is a rather simple system with a very small interface regarding the number of physical connectors. Looking at more involved power plant components like a drum model with multiple fluid connections across the boundary the graphical representation of the FMU holds almost no benefits because it becomes impossible to grasp the purpose of the model at first glance.

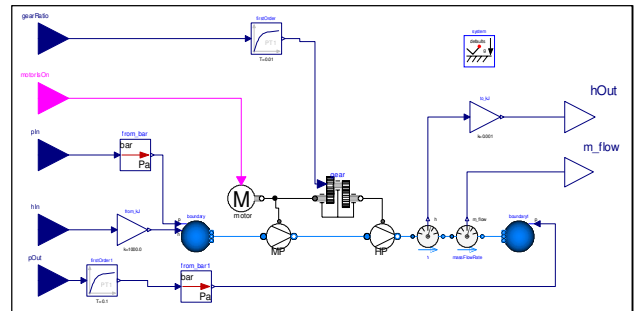


Figure 2: Modelica model of pump system

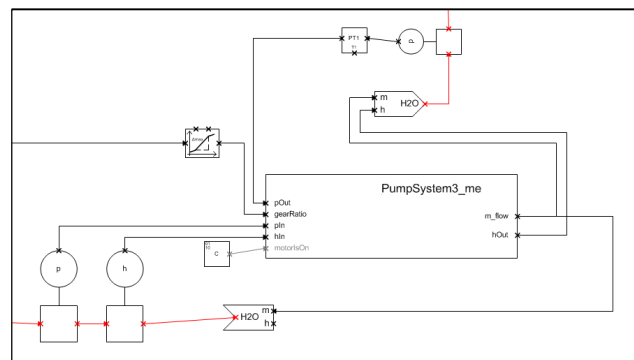


Figure 3: Integration of pump system FMU in Dynaplant

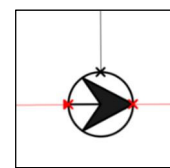


Figure 4: Dynaplant built-in model of a pump system

Besides the loss of information caused by the unstructured signal interface modeling, the lost structure of parameters in an FMU is the most severe shortcoming for us. Particularly, FMUs based on Modelica models make use of parameter hierarchies on many levels. Additionally, almost all professional Modelica libraries use features like grouping and tabs to generate a convenient user interface. Once exported to an FMU all this information is lost and the user is confronted with an unsorted list of parameters.

2.1.2 Controller in Dynaplant (based on generated Modelica code)

For testing plant control systems, Modelica models are automatically generated based on proprietary controller descriptions (Link et. al., 2014). In order to use these controller models in Dynaplant, Modelica models can be exported as FMU and imported into Dynaplant. Tests showed that large controller models are difficult to handle as FMUs for two reasons.

First, they use a bus connector (expandable connector) for signal exchange in Modelica. This bus connector contains hundreds of variables. Figure 5 sketches the concept for signal exchange. A global name space is replicated by using an expandable connector as an inner/outer component. Therefore, actuators and measurements can be placed in the plant on any level in the hierarchy. Furthermore, all boundary conditions are set on the bus connector. This graphical representation of the tested system as well as the underlying control layout is lost when using FMI.

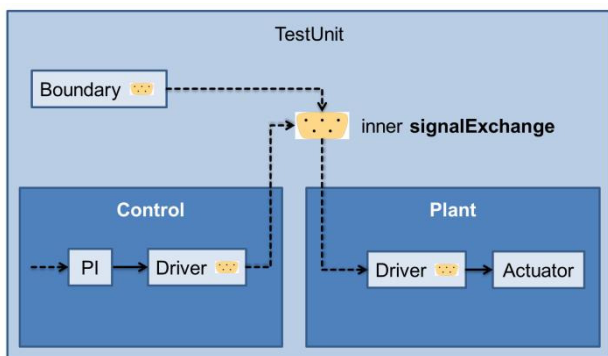


Figure 5: Signal Exchange in Modelica

All exchanged signals need to be available upon FMU import in Dynaplant in order to investigate control behavior. Currently, only scalar connectors are supported by FMI, which makes it hard to handle large sets of bus connector signals after importing the FMU. Future versions of the FMI specification might eliminate this problem.

Second, the controller models include many instances with a huge number of parameters, which are exported into the FMU. The resulting modelDescription.xml measures more than 150 MB. Compared to the Modelica implementation measuring about 6.7 MB (total model with all classes), this leads to slower model import and instantiation. As the compressed FMU appears to be small for the end user, potential causes for performance issues are not transparent. So far, we do not have a solution for providing a compact FMU interface with all required information. Limiting the number of visible parameters on the Modelica side cannot be intended, as they might be useful for investigating control behavior.

2.2 Offline Test of NMPC Loop

This section shows the usage of FMI for offline test of a Nonlinear Model Predictive Control (NMPC) loop. The basic concept of NMPC is to use a dynamic model to forecast system behavior and optimize the forecast in producing 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, resulting in the need for a state estimation algorithm to determine the initial states under consideration of the past record of measurements.

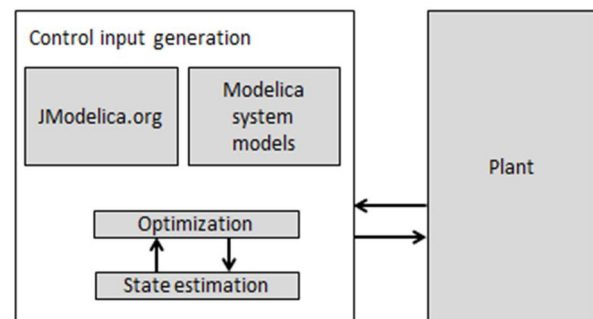


Figure 6: NMPC Loop

The model-based optimization framework looks as follows (see Figure 6):

- Modelica system models are used to describe the dynamics of the process and are used for optimization and state estimation
- The optimization is solved using JModelica.org, the open source platform for optimization, simulation and analysis of complex dynamic systems. JModelica.org interfaces the numerical solver IPOPT and CasADi – the framework for efficient evaluation of expressions and their derivatives
- For online application the optimal control signal is applied to the plant using the OPC interface of the Siemens control system SPPA-T3000 that performs all power plant automation tasks
- For offline tests pseudo measurement data is used to estimate the current state of the model. These measurements are generated by simulating a detailed model of the power plant including the control structure provided as co-simulation FMU exported from Dynaplant

The FMU has an internal state consisting of all values that are necessary to continue a simulation. This

feature could be used for NMPC to restart simulations after initialization with different values of the input signals. Unfortunately, the FMI 2.0 capability *canGetAndSetFMUState* is not yet implemented in JModelica.org. A workaround was implemented to set a consistent internal state, consisting of the values of continuous-time states as well as the iteration variables. It is worth noting that missing FMI capabilities or incomplete implementation is not the exception, but the rule. For each and every intended use case it is up to the user to test the capabilities of the chosen tool.

2.3 ControlTest in Modelica

This last use case is similar to the one from section 2.1.2, with the notable difference that not only control code is available in Modelica but also the plant model.

It is intended to show that the openness and flexibility of native Modelica models should not be underestimated compared to FMI. FMUs, as a translated and compiled version of the model, have well-known drawbacks e.g. when changing interface definitions or trying to understand the internal hierarchical model structure.

On a first look, coupling real-world controller code with physical plant models is a classic use case for FMI. Generally, there is a clear interface between the continuous plant model and the discrete controller implementation.

However, for testing power plant control systems, we currently prefer to have the full system in one uniform Modelica model. Although this requires transferring the graphical structure of control diagrams as well as the full implementation of all control blocks to Modelica. More details on this approach can be found in (Link et al., 2014).

For each study, specific plant models are built, specifically designed for the scope of the control task to be tested. Before starting the first simulations, a considerable amount of work goes into finding consistent boundary conditions for the relevant parts of the plant.

Having one homogeneous test unit in Modelica revealed the following benefits. The interface between plant and controller is not statically defined at the beginning. This allows fast adaptations to new controller strategies. Changes in controller or plant model do not require exporting new FMUs. Even though exporting FMUs requires little effort, ensuring to have the right FMU at the right place, creates unnecessary overhead during model development.

One holistic Modelica model allows control engineers and plant engineers to look into the same model. They both have access to the full system structure, equations and model-integrated documentation. This helps both sides to understand the system behavior and therefore to trust on the results.

This reflects the special situation of usage of FMI in in-house applications, as already mentioned above.

Regarding documentation, most real-world FMUs currently lack on this part (e.g. documentation of model limitations and expected combinations of parameters and input signals). The FMI specification allows documentation to be added by the exporting tool. But, when exporting, it is not guaranteed that the documentation will be provided to user on the importing side. One needs to establish a process on how and where to store the binary FMUs together with the model source code. In contrast, using Modelica, we have one single source of truth, being the Modelica code in version control with hierarchical, model-integrated documentation.

The mentioned drawbacks of FMI are characteristically for any model exchange interface, therefore we do not propose to fix the FMI standard. Instead, we want to encourage users to carefully investigate a pure Modelica solution before introducing FMI interfaces in the tool-chain.

3 Implementation of FMI Support in Dynaplant

With respect to FMI, we support version 1.0 and 2.0 of model exchange import as well as 1.0 of co-simulation export. Implementation efforts are hard to estimate as they naturally depend e.g. on the experience of the implementer and also on the amount of optional features that shall be supported. However, we want to describe briefly in the following the main steps and issues of FMI implementation that we experienced in Dynaplant.

3.1 Import FMI for Model Exchange 1.0

Using an FMU in our tool requires that it can be handled almost as any other component. This means that it has an icon that can be dragged and dropped from a library onto the plant view. Furthermore, the component's inputs / outputs do have graphical port representations that can be connected to ports of other components. Moreover, a parameter dialog needs to be available.

In order to support the described user experience, a gray box component has been added to the component library. After instantiation it shows a parameter dialog for specifying a FMU zip archive. The *modelDescription.xml* is parsed in C# where we generated a class from the available xsd scheme for deserialization. Parameters are then known if revealed by the FMU and it is possible to draw ports and the final component shape. FMUs in version 1.0 and 2.0 do not transport graphical information. Thus, own arrangements have to be done. In our software components usually have predefined ports and shapes and it required significant effort to build the final

component shape only during plant editing. However, after this specification step no extra effort is needed during plant editing for an FMU component, compared to any other component in the plant.

Stepping into simulation, we need to parse the xml-File again in C++, as much more static model information (e.g. information on internals and states) is required during run-time. It did not seem reasonable to overload already our model file (the plant definition) with all information and transfer it to C++. The implementation of calling the FMU functions with the correct arguments and in the right order was manageable with the help of the FMI specification and the FMU SDK of QTronic in mind. As FMUs of version 1.0 cannot transport information on the Jacobian, entries have to be calculated numerically which is quite extensive.

As far as our experience goes, a first implementation of model import can be set up rather easily, but testing and bug fixing is quite time consuming. It is often hindered by tool specific problems like license issues with the exporting tool (even if all partners have valid licenses) or unreasonable start values for inputs in the modelDescription.xml (e.g. some tools give only 0.0 for doubles). These values should be such that they allow for a successful and useful initialization if actually used and not overwritten during the import. Moreover, a general issue in testing is given by the nature of FMI as it is not possible to debug into the FMU to better understand what is happening inside a function call.

3.2 Import FMI for Model Exchange 2.0

Adopting FMI 2.0 has been accomplished by updating the implementation for FMI 1.0 which has been described in the section 3.1. The main effort consisted of updating of the XML parser and of our library of associated convenience functions in C++.

We now support an alternative way of calculating the Jacobian in case the capability flag *providesDirectionalDerivatives* is set to true. We need to provide also an alternative treatment in case an FMU does not support this capability. Generally, the FMI 2.0 specification and also implementation is complicated due to the large number of optional features, in particular capability flags. Two FMUs of version 2.0 and possibly from the same exporting tool can actually be very different in scope and capability. This becomes apparent only in the specific modelDescription.xml files.

With respect to performance, some first tests were done in comparing the import of the same Dymola model as FMU of version 1.0 and 2.0 (Dymola 2016 including bug fix on directional derivatives and sampling). Using the numerical calculation of the Jacobian entries, we experienced a significant decrease

in performance between 1.0 and 2.0. For a rather small example with 92 algebraic and 1322 differential equations in total and 4500 s simulation time we measured a +11 % time usage in the DAE solver. Astonishingly, the loss of time was mainly in Jacobian calculation, although the FMU contributes no states but 310 events. Using the directional derivatives and the associated sparsity information, the performance decreased even further by roughly the same amount. For another example which actually contributed states, the performance of 1.0 and 2.0 roughly leveled up when using directional derivative information which proved to be a benefit in 2.0. Up to this point, a lot of questions remain and further tests are clearly indicated with various types and sizes of models and different exporting tools. It is not fully clear if import or export generate the issues.

3.3 Export FMI for Co-Simulation 1.0

Probably due to the different nature of weak and strong coupling in general, the effort to implement co-simulation export in our tools was significantly smaller than for model exchange import.

Most of the work was consumed in revealing plant information on all inputs / outputs, internals and parameters in a suiting way and offer access via *fmiGet...* and *fmiSet...* functions. The compliance checker was very helpful in bringing the modelDescription.xml in a correct way. It is quite remarkable that our model file of a Modelica similar format consumes roughly more than a factor of 8 less in storage than the modelDescription.xml including only inputs/outputs and parameters but no internals. We derived the learning that it makes sense to look into the FMI ticket trac for recognized issues in the specification, e.g., to find out that some importing tools expect the path to unzipped fmu folders in the call of *fmiInstantiateSlave* whereas others expect it to the zipped path, for instance.

We found testing and bug fixing very difficult and time-consuming also for co-simulation export. As Dynaplant is an in-house simulation tool we do not take part in official FMI cross-checks which could give a hint on still existing bugs but would probably not provide significant details on the root of the issues. It is often hard to track if issues occur in the FMU export implementation or in the importing environment. For the latter, the source code is usually unavailable and debugging possibilities or deeper knowledge on its communication handling are missing.

An important point in our internal discussions is the required resources of a co-simulation FMU. It is necessary to not only transfer one dll but around 35 dlls which are required by our simulator. To begin with, it is not clear where these should be copied to upon import such that the importing environment can

find them because all but one are loaded only implicitly. Any copy requires administrator rights at the target location and upon import, an FMU itself only knows target locations relative to its own unzipped location or the application path. If the latter is chosen, serious problems can occur if dlls with an identical naming can be found there already, potentially of different versions. Moreover, we face some difficulties in unloading all dlls from the address space after a run. The FMI description does not give guidance in dealing with such questions.

4 Summary

Even if the implementation of FMI support in in-house simulation tools implies a great effort, it is useful for many different applications as shown for some use cases in this paper. In principle, FMI can help or even enable some of the shown examples and some of our target applications, but still we face several limitations.

The use cases of chapter 2.1 highlight the urgent need to further develop the FMI standard with respect to the interfaces of FMUs. The existing scalar signal interface is definitely not powerful enough to allow the convenient application of FMI. Either it is to allow the implementation of “physical” connectors as needed in the use case described in chapter 2.1.1. Or to allow structuring of a huge number of signals as described in chapter 2.1.2.

A future FMI standard perfectly suited to our in-house applications would also need to support the concepts of acausal modeling - similar to the built-in behavior of Dynaplan and the basic principles of Modelica. Moreover, we face a mismatch between the intention of FMI to hide information and a need to reveal as much information as possible for in-house application.

Acknowledgements

The work for this paper was partially funded by the German Ministry BMBF (BMBF funding code 01IS12022A) within the ITEA2 project MODRIO.

References

- Franke, J., Brückner, J. (2008): Dealing with tube cracking at Herdecke and Hamm-Uentrop, *Modern Power Systems*, October 2008.
- Christian Bertsch, Elmar Ahle, Ulrich Schulmeister (2014): The Functional Mockup Interface - seen from an industrial perspective, *Proceedings of the 10th International Modelica Conference*, March 10-12, 2014, Lund, Sweden. doi:10.3384/ecp1409627
- Kilian Link, Leo Gall, Julien Bonifay, Matthias Buggert (2014): Testing Power Plant Control Systems in Modelica, *Proceedings of the 10th International Modelica Conference*, March 10-12, 2014, Lund, Sweden. doi:10.3384/ecp140961067

Yongqi Sun, Stephanie Vogel, Haiko Steuer (2011): Combining Advantages of Specialized Simulation Tools and Modelica Models using Functional Mock-up Interface (FMI), *Proceedings of the 8th International Modelica Conference*, March 20th-22nd, Technical University, Dresden, Germany. doi:10.3384/ecp11063491

T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olßon, J.-V. Peetz, S. Wolf, C. Clauß (2011): The Functional Mockup Interface for Tool independent Exchange of Simulation Models, *Proceedings of the 8th International Modelica Conference*, March 20th-22nd, Technical University, Dresden, Germany. doi:10.3384/ecp11063105

A Novel Proposal on how to Parameterize Models in Dymola Utilizing External Files under Consideration of a Subsequent Model Export using the Functional Mock-Up Interface

Thomas Schmitt¹ Markus Andres¹ Stephan Ziegler¹ Stephan Diehl¹

¹3DS GmbH, Germany, {thomas.schmitt, markus.andres, stephan.ziegler, stephan.diehl}@3ds.com

Abstract

This paper introduces a novel proposal on how to parameterize models with data, taking a subsequent model export into account, using e.g. the Functional Mock-Up Interface (FMI). During model export parameters are either assigned with values directly or they are linked to external data-files. If the design of models or libraries is done without considering how data is handled in an exported model, those concepts are often mixed, resulting in an inconsistent data management which is cumbersome or even error prone for the user.

Keywords: model parameterization, data files, model export, functional mock-up interface, FMI, FMU

1 Introduction

In 2011 a new model export standard was released by Modelisar: The Functional Mock-Up Interface (FMI) (Blochwitz et al., 2011), (Association, 2015). FMI was immediately accepted and promoted by many tool vendors and Original Equipment Manufacturers (OEMs). Unfortunately, there are a couple of known pitfalls related to the export of models (Bertsch et al., 2014). One especially relevant for an a-causal modeling language like Modelica is related to the change of an a-causal to a causal model. This required adaption can cause higher index problems and/or algebraic loops (Blochwitz et al., 2012). However, this paper shall deal with a topic not yet intensely discussed by the Modelica community but of central importance for industrial use cases: Parameterization of models, considering a subsequent model export and the handling of data in this case.

From our experience library developers should put considerable effort into proper model parameterization when it comes to a subsequent model export. Fortunately, the Modelica language offers several possibilities to parameterize a model, i.e. to assign parameters with values.

In Modelica it is common to specify parameter values in records. The parameterization can either be done

by coding values into the record with the Modelica environment or by reading the data from an external file for which the format can vary. Both solutions have their pros and cons and are absolutely justifiable. (Köhler and Banerjee, 2005) shows a case where custom text-based files are used as parameter files, which can be accessed by multiple simulation environments. On the other hand, Modelica-based parameter files (records) are usually more convenient for the user, especially for beginners as they can be edited directly in the Modelica environment.

1.1 Use-Cases of Exported Models

In this paper we will focus on the export of FMUs from Dymola¹, discussing different use-cases in which the FMU is utilized after the export. Depending on the particular use-case the model export underlies different requirements regarding convenient data handling. To our experience the following cases cover most of the applications used in industry today.

1. Parameter values are stored inside the FMU.
2. Parameters are stored in an external data-file. The FMU reads the parameter values during initialization of the simulation.
3. The data-file is stored inside the FMU's `resources` folder, i.e. the FMU reads the parameters during initialization, but no external files are necessary.

Each of those use-cases requires a different implementation in terms of model parameterization.

1.2 User Convenience

From our experience it turns out that enabling all three uses-cases significantly enhances the flexibility of the designed models and especially its exported version e.g. an

¹Although other ways of exporting like using the `dymosim.exe` or exported source code should behave the same way.

FMU. This enables the same models to be applied as a complete unit coupling models and parameters (use-case 1), as well as using a single model in multiple applications varying parameters by simply replacing data on a file system² (use-case 2) or enabling a combination of both (use-case 3). Although covering all use-cases would be the most satisfying solution, it is also valuable to decide for the single most important use-case and implement this one throughout the whole library.

Therefore we are very pleased to present a first proposal within this paper. Until today the presented approach is restricted to scalars and tables up to a dimension of two, but an extension to higher dimensions seems reasonable.

2 A Small Modelica Library

For a better understanding the parameterization of the models and the subsequent model export will be demonstrated using an exemplary Modelica library: The TableBasedDiodes library (refer to Figure 1).

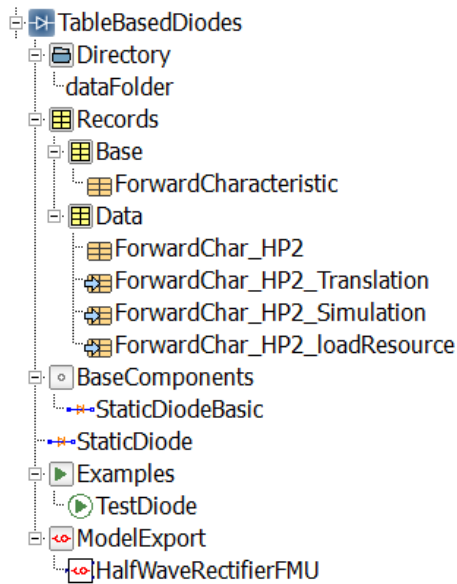


Figure 1: Package structure of the Modelica library

The packages and the models will be explained in the following sections.

2.1 The Utilized Model

The model illustrated in Figure 2, which is located in BaseComponents.StaticDiodeBasic, will be used to demonstrate both the parameterization and the export of the model.

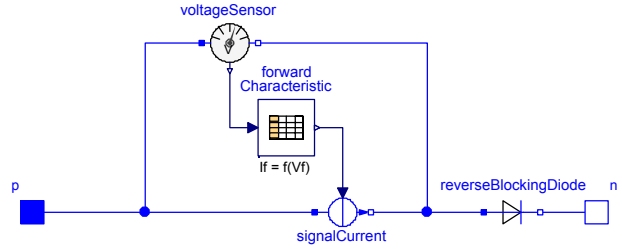


Figure 2: Static model of a diode: $I_f = f(V_f)$

2.2 Model Parameters

To parameterize the model, the diode's forward characteristic, i.e. $I_f = f(V_f)$ is implemented using the one-dimensional table of the MSL (CombiTable1D). Since the reverse characteristic is not always provided in datasheets an additional reverse blocking diode (ideal diode of the MSL) is used to ensure that no current will flow in reverse direction. The parameters of this diode are two scalars describing the on-state resistance R_{on} and the off-state conductance G_{off} .

2.3 Record Structure

To provide different data sets, the parameters are usually declared in records. We propose to provide a partial record, i.e. Records.Base.-ForwardCharacteristic that contains the parameter declarations. This (base) record shall then be extended to assign the parameters with values via modifiers.

Listing 1: Base record of the static diode model

```

partial record ForwardCharacteristic
  extends Modelica.Icons.Record;
  import SI = Modelica.SIunits;

  parameter String Filename = "noFile";
  parameter Boolean tableOnFile = false;

  parameter SI.Current forward[:, :] = fill(0.0
    , 0, 2) "diode's forward characteristic
    i = f(v)";
  parameter SI.Resistance Ron(min = 0, start =
    1e-5) "closed diode resistance";
  parameter SI.Conductance Goff(min = 0, start
    = 1e-5) "opened diode conductance";
end ForwardCharacteristic;
    
```

2.4 Combining Model and Data

A composition of the diode model shown in Figure 2 and the record in Listing 1 will result in the model depicted in Figure 3.

The parameters of BaseComponents.StaticDiodeBasic will be assigned with values stored in the data record via dot-notation. The corresponding code is illustrated in Listing 2.

²Or alternatively modifying the string pointing to the file.

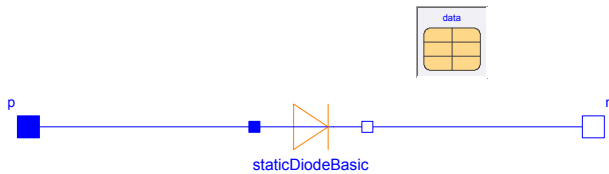


Figure 3: Model and data are combined

Listing 2: Text layer of the static diode model shown in Figure 3

```

model StaticDiode
...
  replaceable
    Records.Base.ForwardCharacteristic data
    annotation(choicesAllMatching);

  BaseComponents.StaticDiodeBasic
    staticDiodeBasic(
      tableOnFile=data.tableOnFile,
      table=data.forward,
      tableName="forwardChar",
      fileName=data.FileName,
      Ron=data.Ron,
      Goff=data.Goff);
...
end StaticDiode;

```

In Figure 3 a partial record is instantiated, i.e. a replaceable model is introduced. If we add the annotation `choicesAllMatching` a drop-down menu appears in the model's parameter window (refer to Figure 4).

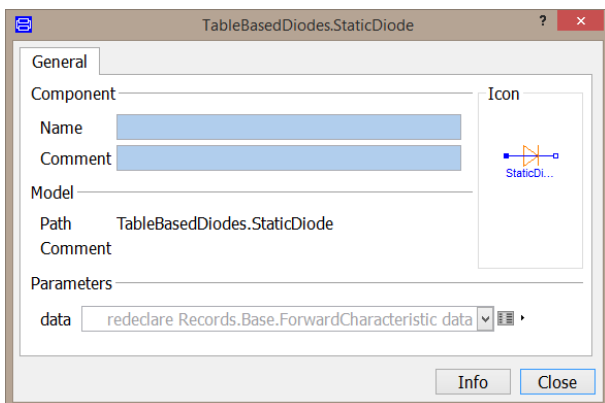


Figure 4: Parameter window of the static diode model

Now, every record that extends the partial (base) record can be selected in the data menu. This will be important to be able to easily distinguish the use-cases.

2.5 Getting Data from Files

If data-files are used, it is possible to influence when the parameters of the model are assigned with values specified in the data-file. Assigning can either happen during model translation³ or at the beginning of the simulation,

³Generating an FMU is a functionality similar to translating a model, resulting in a lot of common parameters.

i.e. during initialization. Assigning parameters during translation means that all values from the data-file are written into the model's code directly. Therefore the file from which the data was read during translation is not needed anymore when the model is simulated. In contrast to that, assigning parameters during initialization of the simulation needs the file available to start the simulation.

Both of the mentioned possibilities can be favorable depending on the scenario the model is used in. Therefore, four implementations will be demonstrated in Section 3 demonstrating how to influence when the parameterization happens. First we will discuss how to handle paths to files efficiently within Modelica.

2.6 File Handling

Typically the data-files are put into a folder located in the libraries root directory, e.g. `TableBasedDiodes` as shown in Figure 1. The data folder is called `Data`. It is common to provide the relative path to this folder inside the Modelica library by introducing, e.g. the package `Directory` shown in Listing 3.

Listing 3: Directory Package

```

package Directory
  constant String dataFolder =
    Modelica.Utilities.Files.loadResource("
      modelica://TableBasedDiodes/") + "Data/"
  ;
end Directory;

```

3 Parameterization of the Model

In the following chapter, four different implementations are introduced to cover the proposed use-cases. Those will be discussed in the following sections.

3.1 Implementation 1: Parameters are Assigned in the Record Directly

A very common case for Modelica library developers is to specify the value of a parameter directly inside the record. This generates an exported model (FMU) that does not need any data-file. This is particularly useful when the user just wants to change only few parameters - preferably scalar values - but not the whole parameter set containing big tables.

Listing 4 illustrates the parameters of the diode's forward characteristic which can be found in the datasheet of Infineon's Hybrid Pack 2.

Listing 4: Data stored in the record

```

record ForwardChar_HP2
  "forward characteristic FS800R07A2E3"
  extends Base.ForwardCharacteristic(
    forward = [0,0; 0.5,0.01;
              0.743,7.749; 1.007,109.203];

```

```

1.126,207.838; 1.226,309.291;
1.306,407.926; 1.379,509.379;
1.440,608.014; 1.502,709.467;
1.555,808.102; 1.609,909.555;
1.663, 1008.190; 1.713,1109.643;
1.755,1208.278; 1.805,1309.731;
1.847, 1408.366; 1.893,1509.82],
Ron = 1e-5,
Goff = 1e-5);
end ForwardChar_HP2;

```

One possible drawback when specifying data directly inside the record is that it is rather inconvenient to modify arrays and matrices. Hence, data is often provided inside data-files, i.e. mat-files, csv-files, hdf5-files or any other file formats or even user-specific file formats.

3.2 Implementation 2: Read Data from File during Model Translation

To read data from csv or mat files Dymola provides functions within the DataFiles package. The function readMATmatrix() needs two arguments, the filename and the variable name stored inside the file. The scalar() function is needed to convert the (1x1)-matrix into a scalar value. In Listing 5 the data is read from the file DiodeFS800R07A2E3.mat. When the model is translated the values of the data-file are stored inside the record, i.e. the data-file is not needed to simulate the model.

Listing 5: Data read from file during model translation

```

record ForwardChar_HP2_Translation
"forward characteristic FS800R07A2E3 from file
during Translation"
extends Base.ForwardCharacteristic(
forward = DataFiles.readMATmatrix(
Directory.dataFolder + "
DiodeFS800R07A2E3.mat", "forwardChar")
,
Ron = scalar(DataFiles.readMATmatrix(
Directory.dataFolder + "
DiodeFS800R07A2E3.mat", "Ron")),
Goff = scalar(DataFiles.readMATmatrix(
Directory.dataFolder + "
DiodeFS800R07A2E3.mat", "Goff")));
end ForwardChar_HP2_Translation;

```

In case of a subsequent model export the exported model will behave exactly the same as the one covered in implementation 1. The major difference is, that in this implementation parameter values are read from a file instead of Modelica code during translation. This can be convenient, as tools specialized on data manipulation can be used to generate the data file and they can be independent of the simulation environment as e.g. in (Köhler and Banerjee, 2005).

3.3 Implementation 3: Read Data from File during Model Initialization

This implementation becomes favorable if the user wants to exchange whole data sets of one and the same model. This can be the case when the user wants to generate only one FMU of a model, using different sets of parameters based on data files.

If the record shown in Listing 5 is modified to the record illustrated in Listing 6 the data will not be saved in the model. This is due to two major differences in the implementation.

For the scalar values Dymola's Modelica compiler assumes that the parameter Filename replacing the constant String in Section 3.2 is intended to be changed, and is therefore kept as a parameter in the compiled model. This makes it possible to change the Filename after model compilation.

For the table values, the ability of the MSL's CombiTables is used, which enables the user to decide if an external file or a table from Dymola shall be used. In this case no internal table is used as shown in Section 3.2, but a reference to a file instead. Therefore the table only receives the path to the file containing the data. During simulation the functions within the table itself will access the data directly from the file specified as Filename. As the data is not written to the model, the size of the tables within the file are not determined during compilation and the sizes of the tables within the datafile can change without modifying the model itself.

Listing 6: Data read from file during simulation

```

record ForwardChar_HP2_Simulation
"forward characteristic FS800R07A2E3 from file
during Simulation"
extends Base.ForwardCharacteristic(
Filename = Directory.dataFolder + "
DiodeFS800R07A2E3.mat",
tableOnFile = true,
Ron = scalar(DataFiles.readMATmatrix(
Filename, "Ron")),
Goff = scalar(DataFiles.readMATmatrix(
Filename, "Goff")));
end ForwardChar_HP2_Simulation;

```

As one can see in Listing 6, the parameter forward (shown in Listing 5) was removed and the parameter tableOnFile was set to true to enable the functionality described above.

String Parameters in Dymola

If the record ForwardChar_HP2_Simulation is chosen (Listing 6), the parameter values are not stored in the FMU. They will be read automatically from the datafile during model initialization. The generated FMU will solely contain the string parameter Filename specifying the path to the data-file. Dymola users have to set the following flag to ensure that string parameters appear inside the FMU:

```
Advanced.AllowStringParameters = true
```

Now one can simply change the to path to the data-file by changing the string parameter.

3.4 Implementation 4: Read Data from File during Model Initialization with Data in the FMU

In many cases it is desired to put the data-files into the FMU since many users don't want to separate model and data when it comes to model export to avoid potential sources of errors. One very common error in that regard is, that data-files are not found as they are not being passed on with the FMU or their (relative) path on the hard drive changed.

Therefore it makes sense to store the data-file in the FMU's resources folder. To do so only a slight modification of implementation 3 (Listing 6) is necessary. The resulting code is shown in (Listing 7).

Listing 7: Record used to store the data-file in the FMU's resources folder

```
record ForwardChar_HP2_loadResource
"forward characteristic FS800R07A2E3 from file
load Resource"
extends Base.ForwardCharacteristic(
  Filename =
  Modelica.Utilities.Files.loadResource
  (Directory.dataFolder + "
  DiodeFS800R07A2E3.mat"),
  tableOnFile = true,
  Ron = scalar(DataFiles.readMATmatrix(
  Filename, "Ron")),
  Goff = scalar(DataFiles.readMATmatrix(
  Filename, "Goff"));
end ForwardChar_HP2_loadResource;
```

The only difference to the record shown in Listing 6 is the loadResource() function applied with the file-name as parameter.

Dymola and the FMU's Resources Folder

To make Dymola copy the file to the FMU, the option *Copy resources to FMU* in the *Dymola Simulation Setup* has to be activated. With this flag set and the loadResource function in use, the resulting FMU will contain a resources folder including the data-file.

As mentioned before this implementation enhances usability, as the user does not have to care about the location of the data files. Still this reduces flexibility as it is not possible anymore to simply change the path to the data file by changing the string parameter introduced in Listing 6. To change the data-file currently the user has to extract the FMU⁴, change the data file and compress it again, which is obviously more effort than just changing a parameter.

⁴Which is just a renamed .zip file.

4 Model Export via FMI

For the model export via FMI we use the model ModelExport.HalfWaveRectifierFMU depicted in Figure 5.

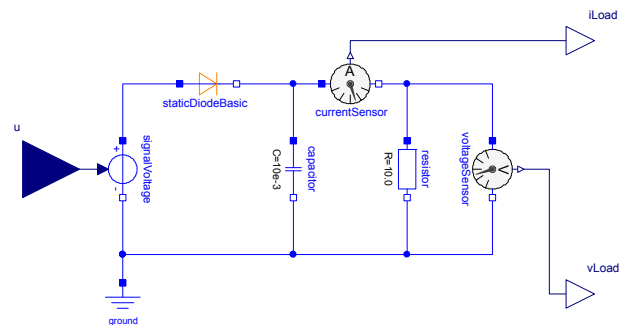


Figure 5: Model used to generate FMUs

We will export the model using the records introduced in the former sections (which can be found in the Records.Data package shown in Figure 1). Changing the record by selecting an entry in the pull-down menu will obviously only change the behavior of the staticDiodeBasic. The parameters of the capacitor and the resistor are not affected by the settings of the diode.

Within the drop-down menu that appears when opening the parameter window of the staticDiodeBasic, one of the four implementations presented in Section 3 can be chosen. Those are:

1. ForwardChar_HP2
2. ForwardChar_HP2_Translation
3. ForwardChar_HP2_Simulation
4. ForwardChar_HP2_loadResource

The identifiers shown in Figure 6 are determined by the model description defined in respective implementations in Listings 4, 5, 6 and 7.

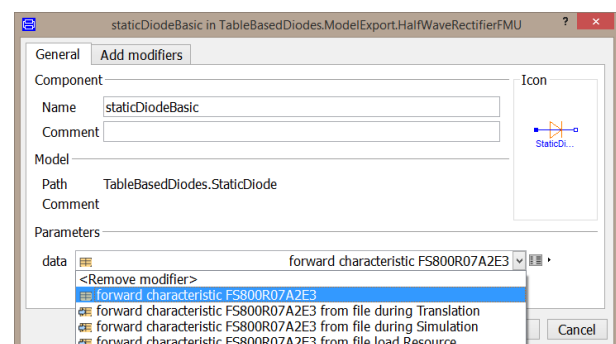


Figure 6: Parameter window of the diode model

By selecting an entry in the pull-down the behavior of both, the model itself in Dymola and the exported FMU

are changed. We can now link the implementations to the use-cases described in Section 1.1.

1. Selecting entry 1 or 2 will both result in a use-case 1 model. The difference will be that entry 1 will rely on data directly stored in the Modelica code and entry 2 will read data from an external file when the model is translated. Still this will not affect the behavior of the generated FMU.
2. Choosing entry 3 will create a use-case 2 model, i.e. load parameters from a data-file which is located in the Data folder with an optional string parameter to change the path to the file (see Section 3.3).
3. Compared to the last point selecting entry 4 will only change the behavior in case of FMU generation representing use-case 3. The data files specified by the `loadResources` function will be copied to the FMU's `resources` folder.

5 Tables with Dimensions Greater than Two

It turns out that the implementation is relatively straightforward for scalar parameters and tables with two independent variables, with simple scalar parameters and tables being based on the MSL's `CombiTables`⁵. Still, today it is not possible - at least with reasonable effort - to provide Modelica libraries covering every use-case as soon as tables with a dimension greater than two are necessary. As tables are of essential importance in the industrial use, some of the problems regarding parameter handling with table-based models will be highlighted now, focusing on an arbitrary number of dimensions.

Implementations of tables with dimensions greater than two (n-dimensional) are provided e.g. by Dymola within the `DataFiles` package `TableND` and by 3DS GmbH's `SimDevTools` (`NDTable`).

However, Dymola's n-dimensional table offers no possibility to enter tables directly or read data from a file during model initialization. Thus, not all use-cases can be covered.

Providing proprietary solutions like the `SimDevTools` fails until today, since Dymola requires to pre-compile functions before the translation of the model (e.g. for the determination of the table size). This has to be done manually until today, resulting in a unacceptable inconvenience for the user.

Mixing different table types implicates a number of disadvantages. Regarding the possible use cases, for a model that includes both, e.g. MSL's `CombiTables` for dimensions up to two and `DataFiles`'s `TableND` for higher dimensions, solely use-case 1 can be covered since the Dymola table reads the values from a data-file

during model translation and in turn stores them inside the model. Additionally those tables behave differently when it comes to interpolation and extrapolation, which is not directly related to model parameterization, but is a major drawback during simulation and debugging. Table 1 shows a summary of the features of the table implementations available today.

In order to enhance table-based modeling which is of central importance in system simulation within an industrial environment, we want to encourage the Modelica community to put even more effort into this topic. Especially into:

- Extending the tables functionality such that it is possible to use data with more than two independent dimensions by default.
- Providing the possibility to import additional data formats, e.g. `hdf5`, ideally in a user-expendable fashion for arbitrary data formats.

6 Compatibility with Other Modelica Simulation Environments

The implementations shown in Section 3 which are required to enable the different use-cases shown in Section 1.1 were created using Dymola and its `DataFiles` package as well as the MSL. Unfortunately, we were not able to test how other Modelica environments (refer to <https://modelica.org/tools>) treat the implementations, which would be important as the resulting behavior is likely to be tool-dependent.

For the Modelica community it would be highly favorable to have a solution like Dymola's `DataFiles` package available in the MSL. This would enable the user to read data from external files⁶ independent of the simulation environment. Ideally it should be extendable to enable customer specific or future data formats.

7 Conclusion

The paper presents three use-cases of model parameterization and four implementations which cover all three use-cases, specifically aimed at a subsequent model export. Additionally it is shown how to implement records with the possibility to choose how an exported model shall behave, by selecting a set of parameters. This property can be set by changing the set of parameters using Modelica based functionality.

This approach is currently only possible for parameters which are scalars, 1D and 2D tables. For higher dimensions it has been pointed out why this is currently not possible.

⁵Which only offer tables up to a dimension of two.

⁶Arrays as well as scalars

Library	Dims	Formats	Interplation	Extrapolation	Data Source	Parametri- zation
MSL CombiTables	1-2	txt, mat, csv (import)	hold, linear, smooth first derivative	linear	Modelica/files	translation, initialization
DataFiles	1-n	mat v4, csv	linear	hold	files	translation
SimDevTools	1-32	sdf (hdf5)	hold, linear, Akima	no, hold, lin- ear	files	initialization

Table 1: Table implementations covered in the paper.

References

- Modelica Association. <http://fmi-standard.org/>, 2015.
- Christian Bertsch, Elmar Ahle, and Ulrich Schulmeister. The functional mockup interface - seen from an industrial perspective. In *Proceedings of the 10th International Modelica Conference*, 2014.
- 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 *Proceedings of the 8th International Modelica Conference*, 2011.
- T. Blochwitz, M. Otter, J. Akesson M., Arnold 4, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International Modelica Conference*, 2012.
- J. Köhler and A. Banerjee. Usage of modelica for transmission simulation in zf. In *Proceedings of the 4th International Modelica Conference*, 2005.

Design choices for thermofluid flow components and systems that are exported as Functional Mockup Units

Michael Wetter¹ Marcus Fuchs² Thierry S. Noudui¹

¹Lawrence Berkeley National Laboratory, Energy Technologies Area, Building Technology and Urban Systems Division, Simulation Research Group, Berkeley CA, USA, {mwetter,tsnoudui}@lbl.gov

²RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Aachen, Germany, mfuchs@eonerc.rwth-aachen.de

Abstract

This paper discusses design decisions for exporting Modelica thermofluid flow components as Functional Mockup Units. The purpose is to provide guidelines that will allow building energy simulation programs and HVAC equipment manufacturers to effectively use FMUs for modeling of HVAC components and systems.

We provide an analysis for direct input-output dependencies of such components and discuss how these dependencies can lead to algebraic loops that are formed when connecting thermofluid flow components. Based on this analysis, we provide recommendations that increase the computing efficiency of such components and systems that are formed by connecting multiple components. We explain what code optimizations are lost when providing thermofluid flow components as FMUs rather than Modelica code. We present an implementation of a package for FMU export of such components, explain the rationale for selecting the connector variables of the FMUs and finally provide computing benchmarks for different design choices. It turns out that selecting temperature rather than specific enthalpy as input and output signals does not lead to a measurable increase in computing time, but selecting nine small FMUs rather than a large FMU increases computing time by 70%.

Keywords: FMI, Modelica, thermofluid flow

1 Introduction

The Functional Mockup Interface Standard (Modelica Association, 2014) is an open standard that has been developed to export models or whole simulators from one simulation software and import them into another simulation software to perform a coupled simulation of time dependent systems. It enables interoperability among simulation software by standardizing (i) an application programming interface and its semantics, (ii) an xml schema that describes the model structure and capabilities, and (iii) the structure of a zip file that is used to package the model, its resources and documentation.

This type of simulation software interoperability is in-

teresting for various use cases in building energy simulation. First, it allows building energy simulation programs, for which it currently is difficult for users to add new models, to add an interface that allow users to insert own component models that may be written in and exported by a variety of simulation software that support the FMI standard (see <https://www.fmi-standard.org/tools> for a list). As a point in case, EnergyPlus currently undergoes a prototype redesign in which HVAC simulation will be based on FMUs (Wetter et al., 2015). Second, the American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) is currently developing Standard 205 that standardizes the representation of HVAC equipment performance data for building energy simulation.¹ As the built-in control and staging algorithms of such equipment affects the performance, participants of the standards committee expressed the need for sharing models as executable code rather than simple performance maps. Here, FMU may be a solution for such model representation. Third, Swegon AB, an international HVAC equipment manufacturer, expressed the need for receiving from their suppliers component models to allow them to optimize the integration of these components into their products. Swegon also is interested in providing equipment models as FMUs to energy simulation programs.

For these use cases, FMI is an interesting technology as it is an open standard that has been designed for the exchange of such models. However, various design questions have to be answered for its effective use, namely: (i) Are both versions of the standard, FMI for model-exchange and FMI for co-simulation, applicable? (ii) If an FMU represents an individual equipment, how would a system simulation program have to execute this component if used as part of an whole HVAC simulation? (iii) What recommendations should one follow to allow an efficient simulation of FMUs if part of an HVAC system simulation? (iv) What code optimization is lost if FMUs are used rather than Modelica, the latter allowing

¹ See <http://spc205.ashraepecs.org/>.

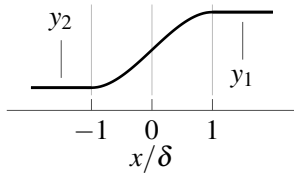


Figure 1. Plot of the function $y = \mathcal{R}(x, y_1, y_2, \delta)$.

symbolic processing prior to code generation. (v) What variables should the parameters, inputs and outputs of an FMU have? Would specific enthalpy as is used in the `Modelica.Fluid` library be a good choice?

This paper addresses the above questions. It is structured as follows: Section 2 states assumptions and introduces notation. In Section 3, we discuss the applicability of the two standards. In Section 4, we provide an analysis of execution sequences and provide recommendations for efficient model implementation. In Section 5, we discuss the design of connector variables. Section 6 explains code optimizations that are no longer possible if FMUs rather than Modelica are used. Section 7 discusses the implementation of FMU export for thermofluid flow components in a development version of the Modelica Buildings library (Wetter et al., 2014), and Section 8 shows numerical benchmarks for different implementations.

2 Terminology and assumptions

2.1 Conventions

1. If a component has two acausal fluid ports, then they are denoted by subscript a and b , respectively, where a is the port into which mass flows under the design flow direction.
2. Consider a model that has an input $u \in \mathbb{R}$, an output $y \in \mathbb{R}^2$ and a parameter $p \in \mathbb{R}$. Suppose $y_1 = pu$ and $dy_2/dt = u$. We say that there is a *direct dependency* between u and y_1 as the value of u needs to be known to produce the output y_1 . In contrast, y_2 does not directly depend on u as it can be produced without knowing the current value of u .

2.2 Notation

To enable robust iterative computation of a numerical approximation to the solution of differential and algebraic systems of equations, the Modelica Specification defines special functions to handle systems in which the flow reverses its direction, and the Modelica Standard Library 3.2 implements regularization functions (Franke et al., 2009; Modelica, 2010). This section explains these functions and the nomenclature that we will use for these functions.

The regularization function `Modelica.Fluid.Utilities.regStep(x, y1, y2, x_small)` approximates

$$y = \begin{cases} y_1, & \text{if } x > 0, \\ y_2, & \text{otherwise,} \end{cases} \quad (1)$$



Figure 2. Connection diagram of three models that is used to explain the concept of stream variables.

by the once continuously differentiable function that is shown in Figure 1. In our discussions, we will denote this function by $y = \mathcal{R}(x, y_1, y_2, \delta)$, with $\delta > 0$. The function is defined as

$$\mathcal{R}(x, y_1, y_2, \delta) = \begin{cases} y_1, & \text{if } x > \delta, \\ y_2, & \text{if } x < -\delta, \\ r(x, y_1, y_2, \delta), & \text{otherwise.} \end{cases} \quad (2)$$

where $r(\cdot, \cdot, \cdot, \cdot)$ is

$$r(x, y_1, y_2, \delta) = \frac{x}{\delta} \left(\left(\frac{x}{\delta} \right)^2 - 3 \right) \frac{y_2 - y_1}{4} + \frac{y_1 + y_2}{2}. \quad (3)$$

Note that some models use `Modelica.Media.Air.MoistAir.Utilities.spliceFunction()` rather than `Modelica.Fluid.Utilities.regStep()`. While these functions are different, their input-output dependency is identical. We will therefore always use the notation $\mathcal{R}(\cdot, \cdot, \cdot, \cdot)$ as our discussion is identical for both implementations.²

To describe in a numerically reliable way the bi-directional transport of specific quantities that are carried by mass flow rate, such as enthalpy, Modelica 3.2 provides the `inStream()` function. Let m_1, m_2 and m_3 be models, and let a and b be fluid ports that are connected as shown in Figure 2. Let h_{outflow} be the specific enthalpy in the connection point if mass leaves the component (regardless of the current flow direction). For the configuration shown in Figure 2, the `inStream()` function satisfies

$$\begin{aligned} \text{inStream}(m_2.a.h_{\text{outflow}}) &= m_1.b.h_{\text{outflow}}; \\ \text{inStream}(m_2.b.h_{\text{outflow}}) &= m_3.a.h_{\text{outflow}}; \end{aligned}$$

In our discussions, we will use the notation $\iota(h_a)$ to denote the value of `inStream(a.h_outflow)`.

2.3 Assumptions

We will make the following assumptions:

1. All components conserve mass, e.g., $\sum_i \dot{m}_i + \Delta \dot{m} = 0$ where the sum is over all ports and $\Delta \dot{m}$ is the moisture added or removed by a humidifier or a cooling coil.
2. Each component has as inputs the mass flow rate \dot{m}_a , the pressure p_a , the temperature $T_{a,i}$ (or specific enthalpy $h_{a,i}$) of the medium that flows into port a if $\dot{m}_a \geq 0$, and the temperature $T_{b,i}$ (or specific enthalpy $h_{b,i}$) of the medium that flows into port b if $\dot{m}_a < 0$.

²In a benchmark for the Annex 60 model library (see <https://github.com/iea-annex60/modelica-annex60/issues/300>) we measured that the function `regStep()` is on average about 8% faster than `spliceFunction()`. Therefore, work is in progress to update the Annex60 and Buildings libraries accordingly.

3. Each component has as outputs the mass flow rate \dot{m}_b , the pressure p_b , the temperature $T_{b,o}$ (or specific enthalpy $h_{b,o}$) of the medium that flows out of port b if $\dot{m}_a \geq 0$, and the temperature $T_{a,o}$ (or specific enthalpy $h_{a,o}$) of the medium that flows out of port a if $\dot{m}_a < 0$.
4. We assume the following direct dependencies: The outlet temperatures are $T_{b,o} = f(T_{a,i}, \dot{m}_a)$ and, similarly, $T_{a,o} = g(T_{b,i}, \dot{m}_a)$ for some functions $f, g: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$.
5. The pressure drop of flow resistances is assumed to be a function of the mass flow rate rather than volume flow rate. The reason for this assumption will become clear in Section 4.

Hence, to simplify the discussion, we typically say that input and output to a component are temperature T . Clearly, models that treat moist air also have water vapor mass fraction X_w as input and output. Except for the discussion of dehumidifying or humidifying components, we do not specifically mention water vapor mass fraction as other components conserve mass. Furthermore the discussion also holds if one were to use specific enthalpy rather than temperature as input and output variables.

3 FMI Standards

FMI 2.0 defines two standards: FMI for model-exchange (FMI-ME) and FMI for co-simulation (FMI-CS): In FMI-ME, the host simulator is responsible for the numerical integration of the model equations, whereas in FMI-CS, the FMU implements its own mechanism for advancing the values of its state variables. FMI-CS provides no mechanism for an FMU to output an instantaneous reaction to a changed input value.³ Hence, FMI-CS cannot be used for steady-state component models of HVAC equipment. However, FMI-ME is applicable. Specifically, FMI-ME allows to set inputs by calling `fmi2SetReal` followed directly by `fmi2GetReal` to obtain outputs. Furthermore, the standard says that `fmi2SetReal` "re-initializes caching of variables that depend on these variables [being set]". Hence, `fmi2SetReal` causes the equations to be evaluated. Therefore, we restrict this discussion to FMI-ME.

4 Direct input-output dependencies of thermofluid flow components and systems

The purpose of this section is to provide guidance to users and developers who connect multiple thermofluid flow component models so they understand when algebraic loops are performed, and how such algebraic loops can be avoided. While in general the existence

of algebraic loops can readily be obtained from the translation information of Modelica tools, the insight we give in this section should inform users and developers *a-priori* about how different component formulations, system compositions and media selections affect the existence of algebraic loops, and how such algebraic loops can be avoided. Based on these discussions, we also provide recommendations for efficient component model formulation.

Questions that this sections answers are:

1. Suppose we know the mass flow rate at each flow leg. Under which arrangements do FMUs, each representing a steady-state fluid flow component, cause an algebraic loop?
2. How does the answer to the above question change if computing the value of the mass flow rate requires solving a flow rate versus pressure drop calculation?
3. Under what conditions does the use of the regularization function to treat near zero mass flow rates cause algebraic loops, and how can they be avoided?

In the next section, we discuss direct input-output dependencies in major HVAC components, and afterwards discuss situations where these components are connected to form HVAC systems.

4.1 Major HVAC components

This section describes the direct input-output dependencies of major HVAC components under the assumption that they are modeled steady-state, as is common in building energy simulation. The purpose of the discussion in this section is to understand what outputs depend directly on what inputs and how direct dependencies can be reduced.

4.1.1 Heater

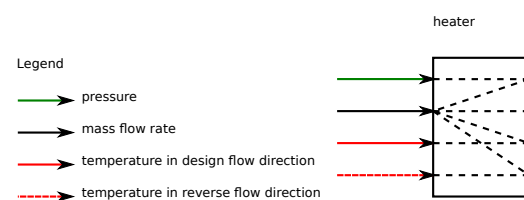


Figure 3. FMU of a heater.

We will start with a simple component of a heater that injects a known amount of heat \dot{Q} into a fluid stream. In such a component, the outlet pressure is $p_b = p_a + f(\dot{m}_a)$ for some function $f: \mathbb{R} \rightarrow \mathbb{R}$, and the outlet temperature is $T_b = g(\dot{m}_a, \iota(T_a))$ for some function $g: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$. For example, for an ideal water heater, $g(\dot{m}_a, \iota(T_a)) = \iota(T_a) + \dot{Q}/(\dot{m}_a c_p)$. We will depict graphically such a component as shown in Figure 3, where the arrows indicate inputs (for this component, inputs are on the left and outputs on the right). The dotted lines inside the component show on what inputs an output directly depends on. We selected to use this graphical representation rather than writing the incidence matrix as the graphical repre-

³ Specifically, FMU-CS does not allow calling `fmi2SetReal` followed by `fmi2GetReal` without first invoking `fmi2DoStep` (see p. 104 of the standard). Furthermore, `fmi2DoStep` does not allow a communication step size of 0.

sensation allows us to graphically connect components to form HVAC systems.

4.1.2 Dehumidifying or humidifying components

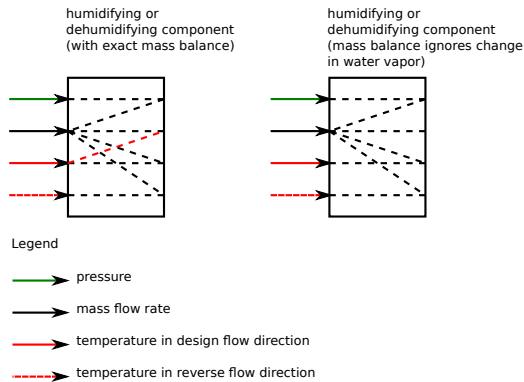


Figure 4. FMU of a humidifying or dehumidifying component. The component on the right implements Recommendation 4.2, and hence the red dashed line is removed.

We now discuss the situation in which the heat exchanger in Figure 3 dehumidifies or humidifies the air. This can be the case for a humidifier or for a cooling coil that cools the air below its dew point. In this situation, the rate of heat and mass transfer affect the outlet mass flow rate. Therefore, if the outlet mass fraction of the humidifying or dehumidifying component depends on the thermodynamic state of the inlet fluid, which generally is the case, then the pressure drop equations are coupled to the heat transfer equations. Hence, such a component has the structure shown in Figure 4. Note that for this component, there generally is no need to properly characterize its thermodynamic behavior for reverse flow because such equipment is only operated when the fan is on. When the fan is off, small reverse flows may occur, but for this situation, it suffices to set $T_a = \iota(T_b)$ and $X_a = \iota(X_b)$, or $T_a = T_{default}$ and $X_a = \iota(X_{default})$, where $T_{default}$ and $X_{default}$ are default values for temperature and mass fraction. The latter version can lead to smaller system of equations if components are used in a flow network.⁴ We therefore decided in Figure 4 that the change in mass flow rate has the functional form $\dot{m}_b - \dot{m}_a = f(\dot{m}_a, \iota(T_a), \iota(X_a))$ rather than $\dot{m}_b - \dot{m}_a = f(\dot{m}_a, \iota(T_a), \iota(X_a), \iota(T_b), \iota(X_b))$. I.e., the thermodynamic properties of the reverse flow are not used to compute the amount of humidification or dehumidification. We therefore make the following recommendation:

Recommendation 4.1 *To reduce the number of direct input-output dependencies of components that humidify or dehumidify the air, such components should implement for the reverse flow `port_a.h_outflow=Medium.h_default`, where `Medium.h_default` is the default specific enthalpy of the medium. Otherwise,*

⁴See <https://github.com/iea-annex60/modelica-annex60/issues/281> for a discussion.

the energy equations for the backward flow become part of the residual functions of the pressure and mass flow rate equations.

Because the outlet mass flow rate is $\dot{m}_b = \dot{m}_a(1 + \Delta X_w)$, where ΔX_w is the change in water vapor mass fraction across the component, this component couples the energy calculation to the pressure drop versus mass flow rate calculations. However, in typical building HVAC systems, $\Delta X_w < 0.005 \text{ kg/kg}$. Hence, by tolerating a relative error of 0.005 in the mass balance, one can decouple these equations. Decoupling these equations avoids having to compute the energy balance of the humidifier and its upstream components when solving for the pressure drop of downstream components⁵. We therefore make the following recommendation:

Recommendation 4.2 *If an error in the mass balance of about 0.5% is acceptable, then one can implement a humidifier or dehumidifier that neglects in the mass balance equation the change in water vapor mass fraction. This can allow computing the mass flow rate versus pressure drop equations without having to couple the energy balance, or the control input of a humidifier or dehumidifier, to these equations.*

As in building simulation, there is considerable uncertainty in air flow rate calculations, and also because larger effects such as duct leakage are generally ignored, taking a relative error of 0.5% into account seems acceptable in typical applications. See also Jorissen et al. (2015) for a discussion.

4.1.3 Fan

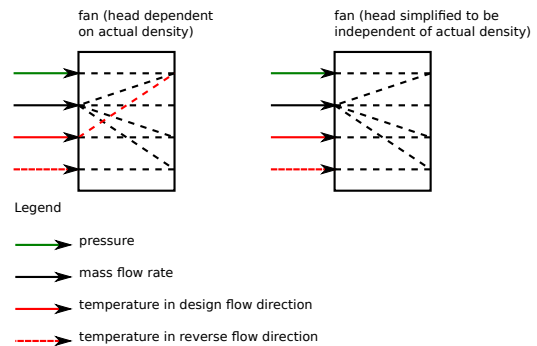


Figure 5. FMU of a fan. The component on the right implements Recommendation 4.3, and hence the red dashed line is removed.

According to the laws of fluid dynamics, the pressure rise over a fan is related to the volume flow rate rather than the mass flow rate. Therefore, the functional form for the fan head is $p_b - p_a = f(\dot{m}_a, \iota(T_a), \iota(X_a))$ and the input-output dependency is as shown in the left-hand side of Figure 5. However, if one were to simplify

⁵In the Buildings library, only downstream components are affected because the humidifier evaluates a component's pressure drop for \dot{m}_a and not for \dot{m}_b .

the fan laws and use a constant mass density, then the direct input-output dependency of the inlet thermodynamic properties could be eliminated, as shown in the right-hand side of Figure 5. We therefore make the following recommendation:

Recommendation 4.3 *If the operating temperature of a fan (or pump) does not change much, or if large uncertainties exist in parameters or the models for the pressure drop calculation of the duct (or pipe) network, then one should assume a constant mass density in the fan model, as this leads to fewer coupled equations.*

4.1.4 Heat exchanger between supply and return air

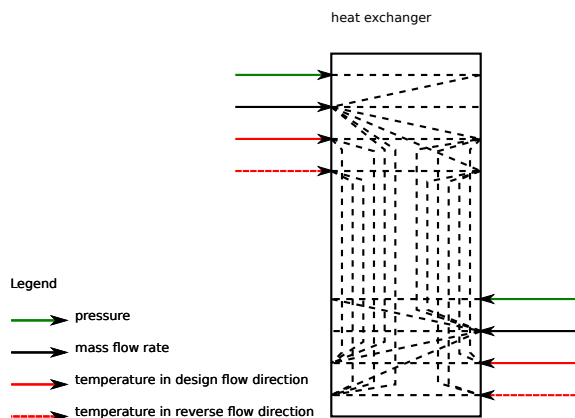


Figure 6. FMU of a component that exchanges heat between two fluid streams.

Figure 6 shows the direct input-output dependency of a heat exchanger. On top left is the inlet of one fluid stream and on the bottom right is the inlet of the other fluid stream. Such heat exchangers are typically modeled using two different implementations.

1. The simplest form is a constant effectiveness heat exchanger. In this situation, the rate of heat transfer is $\dot{Q} = \varepsilon \dot{C}_{min} (T_{in,1} - T_{in,2})$, where $\varepsilon \in (0, 1)$ is a constant, $\dot{C}_{min} = \min(|\dot{m}_{a,1} c_{p,1}|, |\dot{m}_{a,2} c_{p,2}|)$ is the minimum heat capacity flow rate and $T_{in,1}$ is the inlet temperature of the fluid 1, which is equal to $\iota(T_{a,1})$ or $\iota(T_{b,1})$. Hence, the in-streaming thermodynamic properties of the forward and reverse flow must be known in order to compute the thermodynamic properties of the out-streaming fluid for forward and reverse flow.
2. A more elaborate model is one that uses the $\varepsilon - NTU$ model. In this situation the same direct input-output dependency is obtained as for the model with constant effectiveness.

This discussion shows that heat exchangers lead to complex direct input-output dependencies. If one were to compromise on not being able to properly compute the heat transfer if one or both streams reverse their direction, then one could simplify the model to the form shown in Figure 7. Here, we changed the model so that

the transferred heat is zero if any of the flows is different from the design flow direction. Initial experiments indeed confirmed that such a simplified implementation leads to smaller systems of coupled equations. We there-

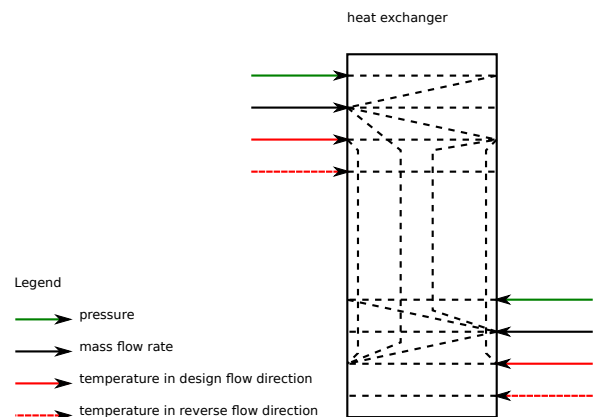


Figure 7. FMU of a component that exchanges heat between two fluid streams but with the simplification that no heat is exchanged if any of the flows is different from the design flow direction.

fore make the following recommendation:

Recommendation 4.4 *If the thermodynamic behavior of a heat exchanger under reverse flow directions is not of interest to the application, then the equations should only be formulated for forward flow. For reverse flow situations, one should simply assign $T_{b,k} = \iota(T_{a,k})$ for both streams $k \in \{1, 2\}$. Note that reverse flow may occur in HVAC systems due physical reasons such as wind pressure on the facade (when the fan is off) or due to numerical artifacts because numerical solvers only compute an approximate numerical solution and hence small negative flows can exist when the HVAC system is off.*

4.1.5 Temperature or humidity control

The user guide `Annex60.Fluid.Sensors.UsersGuide` and of libraries that use Annex60, such as `AixLib`, `Buildings`, `BuildingSystems` and `IDEAS`, recommend to measure temperature, relative humidity, mass fraction, trace substances and specific enthalpy with a sensor that has two ports, and use a dynamic balance to compute the measured quantity. This dynamic balance has shown to be beneficial in large systems that can have zero flow rate. If such sensors are used as an FMU, they have the advantage that the dynamic balance causes the measured quantity to be a state variable. Hence, if used in combination with a P or PI controller, the use of this state variable avoids having to solve an algebraic loop.

Therefore, in the subsequent discussion, we will assume that a dynamic sensor is used.

4.2 Components in series

Figure 8 shows four FMUs in series. This represents the case where a mass flow rate of outside air is conditioned and transported to a room that has a dynamic

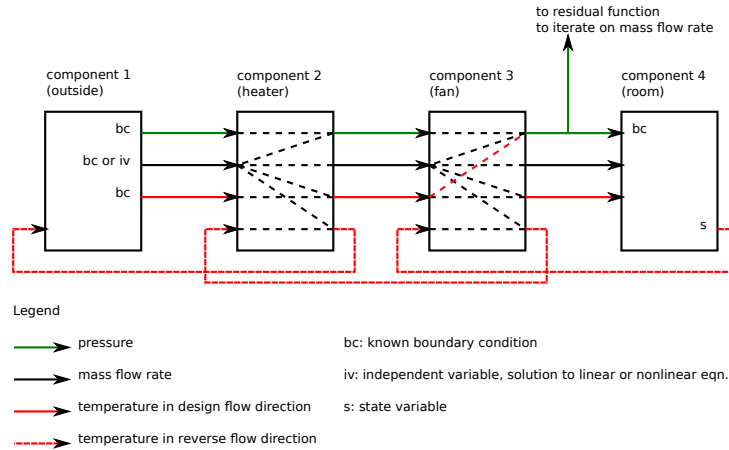


Figure 8. FMUs connected in series.

energy balance, and hence the room air temperature is a state variable whose value is determined by an integration algorithm. The outside air imposes a pressure and temperature boundary condition. The outdoor mass flow rate is either an independent variable or a boundary condition. In the first case, the outdoor mass flow rate is iterated upon until the pressure equations are satisfied. In the second case, the pressure drop equations could be removed from the set of equations.

The equations for this arrangement can be solved as follows. First, by assumption, the pressure drop only depends on the mass flow rate and not on temperature. Therefore, the mass flow rate can be solved iteratively by setting an initial value, evaluating the pressure drop equations of component 1, 2, 3 and 4 in series until a convergence criteria on the difference between the outlet pressure of component 3 and the room air pressure (component 4) is met. Once the mass flow rate is known, components 1, 2, 3 and 4 can be called in sequence to obtain the temperatures for the forward flow direction. For the reverse flow direction, components 4, 3, 2 and 1 need to be called in sequence.

The red line in the fan of Figure 8 is only present if Recommendation 4.3 is not implemented. In this situation, the energy equation of the heater need to be evaluated in order to compute the mass flow rate, thereby increasing the number of operations required to evaluate the residual function.

Analyzing Figure 8 leads us to the following remark:

Remark 4.1 *Evaluating the energy equations for forward flow and then for backward flow is only possible if the energy equations only depend on the thermodynamic state of the inflowing medium. For example, if a component were to use the regularization*

```
h_in = spliceFunction(
    pos    = inStream(port_a.h_outflow),
    neg    = inStream(port_b.h_outflow),
    x      = m_flow,
    deltax = m_flow_nominal/100)
```

then the thermodynamic properties of the backward flow must be known to compute the thermodynamic properties

of the forward flow. Moreover, if, in Figure 8, components 2 and 3 both use the above spliceFunction, then a nonlinear equation must be solved to compute the thermodynamic properties.

Hence, we make the following recommendation:

Recommendation 4.5 *Regularization in which the arguments of the regularization function directly depend on the thermodynamic properties of the forward and reverse flow should be avoided as this can lead to nonlinear equations.*

Note, however, the following:

Remark 4.2 *Simply replacing*

```
h_in = spliceFunction(
    pos    = inStream(port_a.h_outflow),
    neg    = inStream(port_b.h_outflow),
    x      = m_flow,
    deltax = m_flow_nominal/100)
```

with

```
h_in = if m_flow >= 0
    then inStream(port_a.h_outflow),
    else inStream(port_b.h_outflow);
```

is not a solution to Recommendation 4.5. In fact, this would also lead to a non-linear equation, but with a discontinuity in the residual equation, which can lead to problems in Newton-Raphson solvers. Rather, one could attempt to set $h_{in} = inStream(port_a.h_{outflow})$ and let the transferred heat go to zero as the mass flow rate approaches zero from above.

4.3 Components in parallel

Figure 9 is as Figure 8 except that it has two rooms, each with a variable air volume (VAV) terminal. The VAV terminals can increase the flow resistance based on a control signal, and possibly provide heating or cooling (here, we assumed no dehumidification at the terminal unit). To implement such a system, a flow splitter is needed. The flow splitter has as an input the split of the mass flow fraction between the two outlet ports. This input is required as otherwise the splitter is underdetermined.

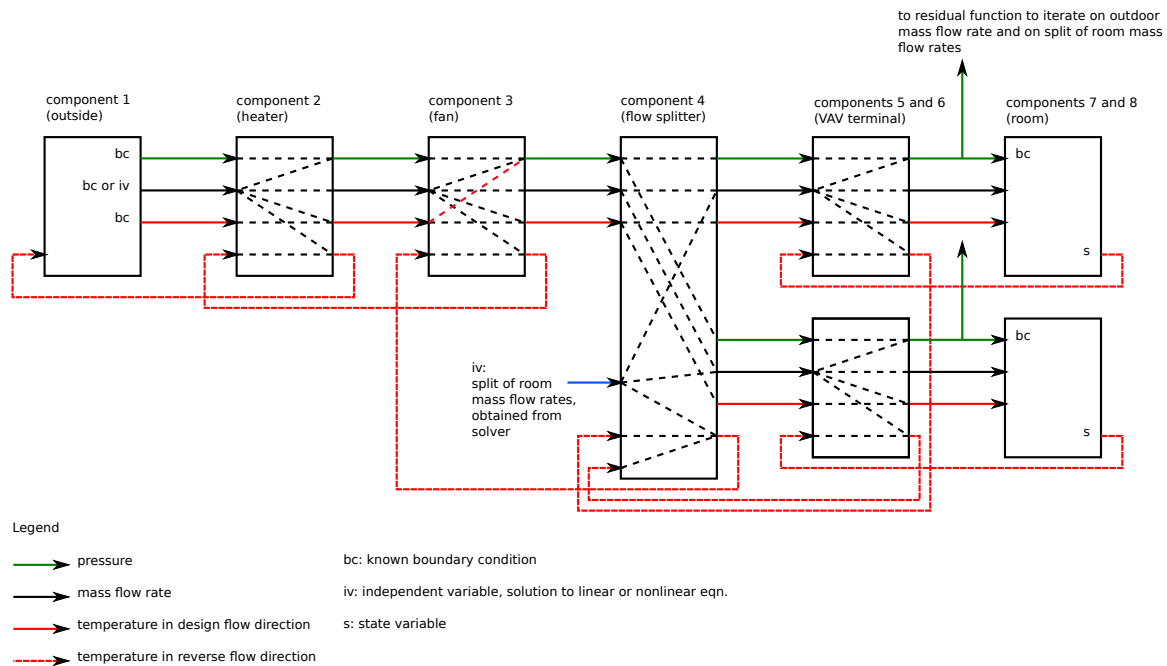


Figure 9. FMUs connected in series that serve two rooms.

The input may be a variable determined by a numerical solver. The VAV terminal has the same input-output relation as the heater.

The equations for this arrangement can be solved as follows: A solver determines the outdoor mass flow rate and the split of the mass flow rates until the residual function of the pressure is satisfied. This can be accomplished by evaluating the pressure drop equations of all FMUs along the flow direction. If the fan does not implement Recommendation 4.3, then the energy equation of components 2 and 3 also need to be evaluated. Once the mass flow rate has converged to its solution, components 1, 2, 3, 4, 5, 6, 7 and 8 can be evaluated for forward flow. Finally, components 7, 8, 5, 6, 4, 3, 2 and 1 can be evaluated for reverse flow.

4.4 Air loops

Figure 10 shows an air loop that consists of the heat exchanger that implements Recommendation 4.4, two fans that implement Recommendation 4.3, and a return duct. The room conserves mass and has a pressure equation for the outlet pressure. Therefore, during the iterative solution of the mass flow rate, convergence is checked on the pressure of the exhaust air.

The equations can be solved as follows: First, components 1, 2, 3, 4, 5, 6 and again 2 are evaluated to solve for the mass flow rate. Next, the energy equation can be solved for forward flow by evaluating components 5 (to get the state T) and 6 to obtain the return air inlet temperature of the heat exchanger. Then, components 1, 2, 3, 4 and 5 can be evaluated, which concludes the computations for the forward flow. For the reverse flow direction, components 1, 5, 4, 3, 2, 6 and again 1 and 5 can be evaluated. Note that the order is not unique as one could have

started with component 5.

Remark 4.3 Note that the heat exchanger is called at least four times if flow reversal is allowed, i.e., twice for the iteration for the mass flow rate, once for forward flow and once for reverse flow. Without flow reversal, the heat exchanger is called at least three times. This indicates the inherent inefficiencies when using FMUs for individual fluid flow components, rather than letting the symbolic processor of a Modelica tool rearrange the equations to a block lower triangular form.

4.5 Control Loops

As discussed in Section 4.1.5, feedback control loops for thermodynamic properties such as temperature or humidity do not cause an algebraic loop if a dynamic sensor is used. Specifically, if a sensor from the package `Buildings.Fluid.Sensor` is used and its time constant `tau` is set to a value larger than zero, then the sensor will output a state variable and hence the feedback control loop does not cause an algebraic loop. If `tau=0` and the controller has direct feedthrough, then such control loops for steady-state HVAC components cause an algebraic loop. To avoid such algebraic loops, the controller could be idealized and implemented directly in the HVAC component, as is done for example in the model `Buildings.Fluid.HeatExchangers.HeaterCooler_T`.

5 Connector variables

In this section, we discuss the selection of the variables that will appear as inputs and outputs of the FMU. We recall that `Modelica.Fluid`, `Annex60.Fluid` and libraries that depend on it such as `AixLib`,

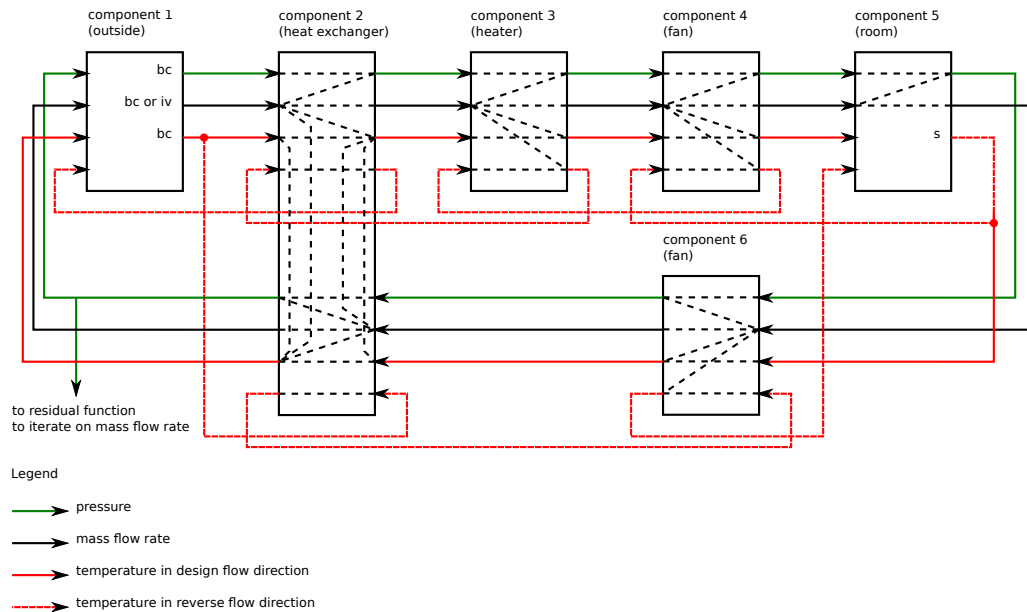


Figure 10. FMUs connected in a loop with a heat exchanger between the outside air intake and the exhaust air.

Buildings, BuildingSystems and IDEAS, use for the pressure the total pressure in Pascal, for the mass flow rate kg/s and for the mass concentration for moist air kg/kg total air. We will use the same variables for the parameters, input and output signals of the FMU. While the connectors in the above libraries use specific enthalpy h , we will use the absolute temperature T in Kelvin instead. The reasons for this selection are as follows:

1. If we were to use the specific enthalpy h as a connector variable, then an FMU would not be self-contained. Rather, to use the FMU, one would require knowledge of the function that is used by the FMU to relate temperature and mass fraction to enthalpy. Consequently, exchanging models in FMUs would not be possible without also providing such a function.
2. In Modelica, using h is motivated as it allows mixing of fluid streams in a port, e.g., in a port, $h_{mix} = \sum_i \max(0, \dot{m}_i) h_i / \sum_i \max(0, \dot{m}_i)$, where h_i is the enthalpy of the fluid that flows into the port. Using temperature in the mixing equations that are generated by the fluid connections can give wrong results as $T_{mix} = \sum_i \max(0, \dot{m}_i) T_i / \sum_i \max(0, \dot{m}_i)$ only holds if the specific heat capacity c_p is constant. However, this is not a concern for FMUs as mixing in ports is supported in Modelica but not when FMUs are connected among each other.

We also had to make a choice about using Kelvin or degree Celsius for the temperature. Whereas users may be more accustomed to use degree Celsius, we decided to use Kelvin for the following reasons:

1. FMUs for model-exchange and for co-simulation not only expose input and output signal, but also state variable and parameters. The state variables in models of the Buildings library are temperature in Kelvin. Changing them to degree Celsius would require re-

designing the library, and hence using a unit convention in the Buildings library that is different from what is used in the Modelica Standard Library.

2. Without such a redesign, FMUs would require some temperatures in Celsius and others in Kelvin.
3. Many models have parameters for design temperatures, and also compute outputs that are temperatures, such as temperature sensors or the temperature of a furnace. These quantities have units of Kelvin. Hence, for all parameters and all such signals, a unit conversion would need to be implemented, which would be quite cumbersome. Moreover, such parameters and variables may still show up as an FMU interface variable, thereby introducing mixed units.

Because using mixed units is confusing and error-prone, we use Kelvin and propose that tools handle unit conversions between the computed quantities and the quantities that are displayed to the user, as is done for example in Dymola 2016.

With these design decisions, an FMU that has two fluid ports called `inlet` and `outlet` will have the following interface variables.

```

inlet.m_flow
  p
  forward.T
    X_w
    C
  backward.T
    X_w
    C
    
```

where `m_flow` is the mass flow rate, `p` is the absolute pressure (which is conditionally removed if `use_p_in=false`) and `forward` and `backward` are the thermodynamic properties for the forward flow and backward flow. If `allowFlowReversal=false`, then `backward` is removed. The thermodynamic vari-

ables are temperature T in Kelvin, water vapor mass fraction X_w in kg/kg total air, which is removed for water, and trace substances C , which is removed if `Medium.nC=0`.

6 Code optimizations lost by using small FMUs

This section describes code optimizations that are no longer done when models are shared as an FMU as opposed to sharing Modelica mode, because FMUs either contain compiled code or C-code, neither of which allows the level of computer algebra possible with Modelica. While the Modelica specification does not prescribe the code optimization, most Modelica compilers are expected to conduct the optimizations described below.

1. Consider Figure 8. A Modelica compiler would use one variable for the mass flow rate that enters the component, and one for the leaving mass flow rate. Results can be written efficiently by storing only the mass flow rate $\dot{m}_{1,b}$ that leaves component 1, and declaring in the output file that $\dot{m}_{k,b} = \dot{m}_{1,b}$ for $k \in \{2,3\}$ and $\dot{m}_{k,a} = \dot{m}_{1,b}$ for $k \in \{2,3,4\}$. However, such knowledge is no longer available if multiple FMUs are used. Hence, mass flow rates must be set, read, stored and written to disk multiple times. Similar discussions apply for thermodynamic properties that remain unchanged in a component, such as T , X_w and C in an air damper.

The efficiency loss that incurs if the output has the same value as the input could, however, be avoided using optional features of the FMI standard. For example, first, if variables share the same `valueReference` in the `modelDescription.xml` file, then they have the same value. Second, if `dependenciesKind="fixed"` is declared in the `modelDescription.xml` file, then the output is, after `fmi2ExitInitializationMode`, equal to a fixed factor times the input, and hence a master algorithm can deduce that they are equal. Dymola 2016 uses the latter construct.

2. Consider Figure 8. If $\dot{m}_{1,b}$ is an iteration variable, components 2 and 3 can be configured to compute pressure drop as a function of the mass flow rate, rather than mass flow rate as a function of the pressure drop, thereby keeping the number of iteration variables as small as possible. Such a selection is no longer possible if a component is exported as an FMU. See also Jorissen et al. (2015) for how this can affect computing time.
3. Consider Figure 8. A Modelica compiler may do automatic differentiation of the Modelica code to compute a symbolic expression of the Jacobian matrix that is used to iteratively solve for the mass flow rate that satisfies the constraint on the pressure.
4. In Figure 9, if the VAV terminals 5 and 6 both require the evaluation of psychometric functions that depend

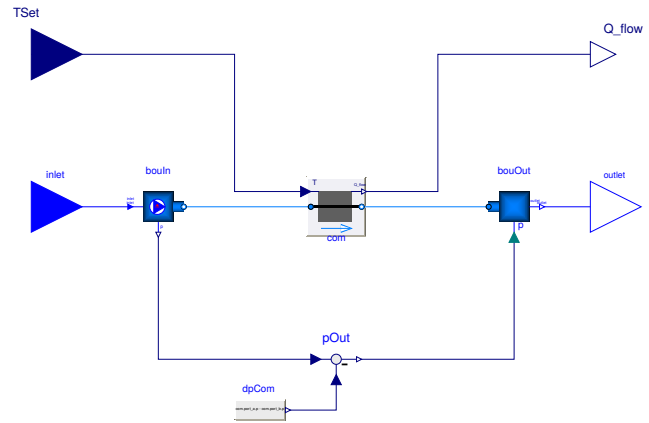


Figure 11. Block that contains a replaceable model of a heater and that defines input and output signals for export as an FMU.

on its inlet temperature and humidity, which are equal for components 5 and 6, then Modelica compilers can compute these functions once and assign the results to both components using what is called common sub-expression evaluation.

5. If no pressure drop calculation is requested, in Modelica it is possible to remove all these computations. In FMUs, while computations can be disabled with a boolean parameter, there will still be an input-output dependency, causing a system simulator to wrongfully believe that there is an algebraic loop.
6. If multiple components form a system of equations, Modelica compilers may solve it explicitly if it is small and linear. If it is nonlinear, a Modelica compiler can use block-lower triangularization and tearing to reduce its dimension (Cellier and Kofman, 2006).

As a drawback when allowing these optimizations, one would require a Modelica translator and a C-compiler on the host simulator. Also, for large Modelica models that involve buildings and HVAC systems, translation and compilation time can be in the order of minutes in tools such as Dymola or OpenModelica. This, however, could be reduced by compiling only the HVAC system, and by doing incremental parallel compilation.

7 Implementation

We implemented the package `Buildings.Fluid.FMI` that contains connectors, blocks that have replaceable thermofluid components, examples blocks that can be exported as FMUs, and examples in which we connected these example blocks to form system models.

Figure 11 shows such an example block. In the middle is the actual thermofluid component. In this case, it is a heater or cooler, but it may be a whole subsystem that contains multiple thermofluid components as long as it extends `Buildings.Fluid.Interfaces.PartialTwoPort`. To the left and right are adaptors that convert between the signal flow and the acausal fluid connectors. At the bottom is the computation of the pressure difference across the component. This is required

as one adaptor needs to set the flow rate and the other the pressure in order for the component to be balanced. The connectors `inlet` and `outlet` contain the input and output signals. The `inlet` connector is defined as follows (most annotations have been removed for better readability):

```
within Buildings.Fluid.FMI.Interfaces;
connector Inlet "Connector for fluid inlet"
  import FMI = Buildings.Fluid.FMI;
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium
    "Medium model";
  parameter Boolean use_p_in = true
    "= true to use a pressure from connector";
  parameter Boolean allowFlowReversal = true
    "= true to allow flow reversal";
  input Medium.MassFlowRate m_flow
    "Mass flow rate into the component";
  FMI.Interfaces.PressureInput p
    if use_p_in
    "Thermodynamic pressure";
  input FMI.Interfaces.FluidProperties
    forward(
      redeclare final package Medium = Medium)
    "Inflowing properties";
  output FMI.Interfaces.FluidProperties
    backward(
      redeclare final package Medium = Medium)
    if allowFlowReversal
    "Outflowing properties";
end Inlet;
```

The connector `Buildings.Fluid.FMI.Interfaces.FluidProperties` contains the thermodynamic properties, and is defined as follows:

```
within Buildings.Fluid.FMI.Interfaces;
connector FluidProperties
  "Type definition for fluid properties"
  import FMI = Buildings.Fluid.FMI;
  replaceable package Medium =
    Modelica.Media.Interfaces.PartialMedium
    "Medium model";
  Medium.SpecificEnthalpy h
    "Specific thermodynamic enthalpy";
  FMI.Interfaces.MassFractionConnector X_w
    if Medium.nXi > 0
    "Water vapor mass fractions per kg total air";
  Medium.ExtraProperty C[Medium.nC]
    "Properties c_i/m";
end FluidProperties;
```

Note that we introduced the new connectors `Buildings.Fluid.FMI.Interfaces.PressureInput` and `Buildings.Fluid.FMI.Interfaces.MassFractionConnector`. The first was required to conditionally remove the pressure from the connector. For example, if a user is not interested in computing the pressure drop, then setting the parameter `use_p_in=false` will eliminate `p` from the connector, remove all pressure drop calculations and setting the pressure of the component to `Medium.p_default`. We also decided to introduce the new connector

```
within Buildings.Fluid.FMI.Interfaces;
connector MassFractionConnector =
  Modelica.SIunits.MassFraction
  "Water vapor mass fraction per kg total mass";
```

to avoid having a vector with one component for the water vapor mass fraction. This was done so that the

FMUs have as an input or output for the water mass fraction a scalar variable `X_w` rather than having a vector with one component for the water vapor mass fraction.

In the `Buildings` library, when running the regression tests, for each model that is exported as an FMU a file will be generated that shows the dependencies of outputs, states and initial unknowns. This file can be inspected to see what dependencies thermofluid flow components have, and the file will be used in subsequent regression tests to verify that the dependencies do not inadvertently change when models are revised.

8 Numerical experiments

8.1 Connector Variables

To benchmark the computing time with temperature T versus specific enthalpy h in the FMU input and output, we simulated an HVAC system. The HVAC system is a VAV system with economizer, heating and cooling coil in the air handler unit, and models of return duct, splitter, terminal heaters and controls. The FMUs either had T or h as input and output variables. Internally, the models use enthalpy balance, and hence if T is an input and output variable, a conversion from T to h is required for the input and from h to T for the output. We exported the components as nine FMUs from Dymola 2015 FD01 and connected and simulated them in Ptolemy II (Ptolemaeus, 2014) for the same number of steps. To bridge from Java used in Ptolemy II to FMI, we used the Java Native Interface (JNI). This experiment did not show a difference in computing time.

Next, we simulated the Modelica implementations with T or h in the inlet and outlet signals, connected to a first order room response, directly in Dymola with the `Rkfix3` integration algorithm, without use of any FMUs. This experiment also showed no difference in computing time.

These two experiments indicate that there is no performance penalty of choosing T rather than h for the input and output signals.

8.2 Code optimizations lost by using small FMUs

To investigate the impact of lost code optimization, we simulated the HVAC model of Section 8.1 that was exported as either one or as nine FMUs. Both systems were simulated in Ptolemy II for 35,040 steps, which would correspond to an annual simulation with an average time step of 15 minutes. The simulation time was 2 seconds for the case with one FMU, and 3.4 seconds for the case with nine FMUs. Hence, using nine FMUs increased the computing time by 70%. The difference is attributed to the lost code optimization in FMUs, the overhead of calling many FMUs, and transferring data between outputs and inputs of FMUs. Note however that the version of Ptolemy II that we used for our experiments does not

make use of the dependenciesKind information explained in Section 6, item 1.

9 Conclusions

The analysis in Section 4 showed that regularization and the use of the `inStream` function can cause direct input-output dependencies in FMUs that contain steady-state HVAC equipment models. Recommendations to avoid such dependencies are provided. We also provided various recommendations to implement approximate equations in thermofluid flow models that lead to fewer input-output dependencies, and hence smaller coupled systems of equations.

Our analysis showed that using multiple small FMUs prevents system-level code optimization that is otherwise done in Modelica. This was confirmed by our numerical experiments.

For users, we provide a Modelica package that allows export of thermofluid flow components and systems for different media, with and without pressure drop calculations.

In summary, the efficiency of using FMUs for thermofluid flow components strongly depends on component design, and various code optimizations are lost when using small FMUs rather than Modelica models.

10 Acknowledgment

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.

This work emerged from the Annex 60 project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme. Annex 60 will develop and demonstrate new generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and Building Information Modeling standards.

References

- François E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer, 2006.
- Rüdiger Franke, Francesco Casella, Martin Otter, Michael Sielemann, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Stream connectors – an extension of modelica for device-oriented modeling of convective transport phenomena. In Francesco Casella, editor, *Proc. of the 7-th International Modelica Conference*, Como, Italy, September 2009. Modelica Association. URL <https://www.modelica.org/events/modelica2009/Proceedings/memorystick/pages/papers/0078/0078.pdf>.
- Filip Jorissen, Michael Wetter, and Lieve Helsen. Simulation speed analysis and improvements of Modelica models for building energy simulation. In *11-th International Modelica Conference*, Paris, France, September 2015. Modelica Association.
- Modelica, 2010. *Modelica – A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 3.2*. Modelica Association, March 2010. URL <https://www.modelica.org/documents/ModelicaSpec32.pdf>.
- Modelica Association. *Functional Mock-up Interface for Model-Exchange and Co-Simulation version 2.0*, 2014. URL <https://www.fmi-standard.org/downloads>.
- Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. URL <http://ptolemy.org/books/Systems>.
- Michael Wetter, Wangda Zuo, Thierry S. Noudui, and Xiufeng Pang. Modelica Buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014. doi:10.1080/19401493.2013.765506.
- Michael Wetter, Thierry S. Noudui, David Lorenzetti, Edward A. Lee, and Amir Roth. Prototyping the next generation energyplus simulation engine. Accepted: *13-th IBPSA Conference*. International Building Performance Simulation Association, December 2015. URL <http://www.ibpsa.org/>.

FMI for physical models on automotive embedded targets

Christian Bertsch¹ Jonathan Neudorfer¹ Elmar Ahle¹
Siva Sankar Arumugham² Karthikeyan Ramachandran² Andreas Thuy³

¹Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, 71272 Renningen, Germany

²Robert Bosch Engineering and Business Solutions Private Limited, 560 103 Bangalore, India

³ETAS GmbH, 70469 Stuttgart, Germany

Christian.Bertsch@de.bosch.com Jonathan.Neudorfer@de.bosch.com Elmar.Ahle@de.bosch.com
SivaSankar.Arumugham@in.bosch.com Karthikeyan.R@in.bosch.com Andreas.Thuy@etas.com

Abstract

This paper explores the possibility to include and execute source code functional mockup units on Bosch electronic control units. A prototypical realization is presented, and assumptions as well as limitations are documented. Special emphasis is laid on requirements for the contained C-code. Furthermore, aspects for an extension to adapt the FMI to the usage on automotive embedded real-time systems are summarized.

Keywords: Automotive, FMI, Embedded Systems, Electronic Control Units

1 Introduction

The Functional Mock-up Interface (FMI) is a well received tool independent approach for model exchange (Blochwitz et al. 2011, 2012) and a promising candidate to become the industry standard for exchange of models and cross-company collaboration (Bertsch et al. 2014). From the beginning in the MODELISAR project, a wide range of possible simulation target platforms for FMI was foreseen, ranging from offline simulation platforms over Hardware-in-the-Loop (HiL) real-time systems to embedded systems (Chombard 2012). While in the offline simulation world FMI is well-established, this is not the case for embedded applications.

This paper presents results of a prototypical FMI implementation for physical models on automotive embedded targets. Furthermore, the usage and adaptation of the FMI as a standard for the automotive embedded world is investigated.

Section 2 provides an overview of the state of the art of physical models on real-time systems. In Section 3, a prototype implementation of FMI on an Electronic Control Unit (ECU) is introduced. Section 4 proposes aspects for an extension to adapt the FMI to the usage on automotive embedded real-time systems before Section 5 concludes with a summary and an outlook.

2 Physical models for real-time systems

First, an overview of real-time systems is given where physical models are already implemented using the FMI standard (Section 2.1). Then, the status on embedded systems is examined (Sections 2.2 and 2.3) and the idea as well as advantages of the FMI on embedded systems is presented in Section 2.4.

2.1 FMI for real-time systems: state of the art

The usage of FMI for real-time simulation is gaining importance. The import of Functional Mock-up Units (FMUs) is supported on major HiL systems in the automotive domain, e.g., ETAS LABCAR (Mitrohin 2014). The same is true for rapid prototyping applications (Brembeck et al. 2014). There are even first applications of FMI for productive online control and optimization tasks of power plants (Franke 2015). All of these real-time applications have one thing in common, i.e., they run on personal computers (PC) or similar systems.

2.2 Embedded systems and ECUs

In contrast to real-time applications running on PCs, software on automotive ECUs has special requirements. Some of them are more generally true for embedded systems, some of them apply only to automotive embedded systems such as special coding guidelines (MISRA 2013) or software architecture (AUTOSAR 2015).

Typically, a modern vehicle has a lot of different ECUs for different tasks such as a vehicle control unit, an electronic stability control unit, and last but not least an engine control unit for internal combustion engines. ECUs are embedded systems with a different design compared to other computer systems such as PCs: Usually they have restrictions on

- memory usage,
- power consumption as well as computational power,
- hard real time requirements, and

- available libraries: Not all mathematical functions known in the offline world, e.g., as declared in `math.h`, are available for embedded targets.

2.3 Physical models on ECUs: state of the art

Physical models play an important role in the advanced control of internal combustion engines. Application fields are, e.g., in the advanced control of the air system of internal combustion engines. Also the real-time capable implementation of advanced numerical methods such as implicit discretization of stiff ordinary differential equations for ECU software functions have been successfully implemented (Wagner et al. 2008) and are used in real-life applications. One major advantage of physical models is that they can reduce calibration effort and the number of characteristic maps (Seuling et al. 2012).

The demand for the implementation of physical models is rapidly growing due to the growing complexity of systems (e.g., hybrid electrical systems), higher demands on control and diagnosis due to efficiency and legislation. There is a need for new model-based methods for

- virtual sensors, i.e., observers,
- model-based diagnosis,
- inverse physical models as feed forward part of control structures, and
- model predictive control.

In many cases the solution of the physical equations can be separated from other of the control algorithms and thus this part could be encapsulated an FMU. An example for this is shown for the model-based diagnosis, as depicted in Figure 1.

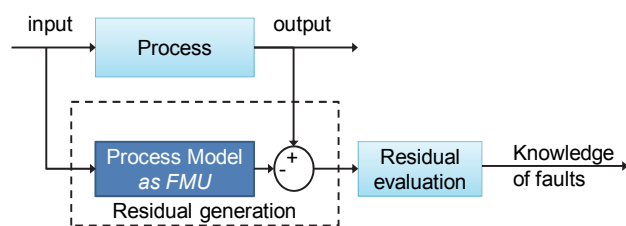


Figure 1: Encapsulated physical model as FMU in model-based diagnosis, adapted from (Ding 2013)

For the offline development and simulation of such physical model, powerful tools are available. However, the generated C-code from the majority of the tools cannot be ported directly to an ECU. Today, these models are often (re-)implemented individually for the specific use case on the ECU. This requires expert knowledge and significant effort.

Furthermore, there is no standardized interface for physical models available on the ECU. Although the AUTOSAR standard is widely accepted in the automotive domain, it is not supported by typical modeling tools for physical models. Also, AUTOSAR

does not provide an interface for solvers of Ordinary Differential Equations (ODEs) and related mathematical description.

2.4 The idea and advantages of FMI on the ECU

The idea of using FMI not only in PC-based environments but also for embedded systems is already considered as a possibility in the existing FMI standard. Our motivation to work in this direction is to facilitate the implementation of physical models on the ECU by defining re-usable interface-functions and numerical algorithms. This enables improvement in the development process for physical models within ECU software: The models could be seamlessly reused from early design (offline), over rapid control prototyping, and then finally ported to the ECU. This will also facilitate the collaborative development of ECU software between original equipment manufacturers (OEMs) and their suppliers in the context of physical models.

There are a lot of tools available for physical modeling that support the FMI standard (also with C-code generation), but only very few that support the AUTOSAR standard. In a first step, one can apply the prototype presented below for rapid (control) prototyping purposes with physical models being exported as FMUs from different tools.

Later – when additional requirements on the included C-code would be fulfilled and the standard would be adapted – it could also be used to exchange code for real ECU projects. The core question is how to integrate the FMU and its numeric framework into an ECU software architecture like AUTOSAR.

3 Prototype implementation of FMI on an ECU

A first prototype for using FMI as an intermediate format in AUTOSAR was described in (Thiele et al. 2011) with the intention of using FMI as a lean standard to exchange software components. There it is pointed out that compared to AUTOSAR, the FMI standard is much smaller and more straightforward, and support of the FMI standard is a more manageable task. Additionally, (Thiele et al. 2011) describes in detail the mapping of FMI and AUTOSAR XML configuration files and interfaces. In our approach this is done in a similar way, as described in Section 3.2.1. However, in contrast to our intention, the emphasis of (Thiele et al. 2011) was on pure control software and on the software architecture. In this paper, the focus is laid on porting physical models to an ECU and actually computing them on the target platform.

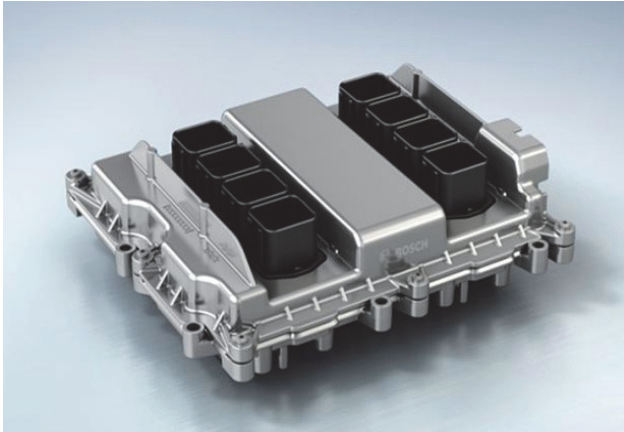


Figure 2: Bosch MDG1 ECU

As a demonstration platform the latest ECU platform of Bosch for powertrain applications is selected - the new “MDG1 ECU” as depicted in Figure 2. It is a scalable multi core processor system, see (Rüger et al. 2013) for details. As shown in Figure 3, this platform supports both application software (ASW) in a Bosch specific format compatible with former Bosch ECU generation “MEDC17”, and AUTOSAR. The base software is fully AUTOSAR compliant and communicates with the AUTOSAR application software via the run-time environment (RTE) and with MEDC17 application software with a Bosch-specific interface.

This is the platform for which the execution of FMUs has been prototypically realized within the real ECU software infrastructure (i.e., the build toolchain and special compilers).

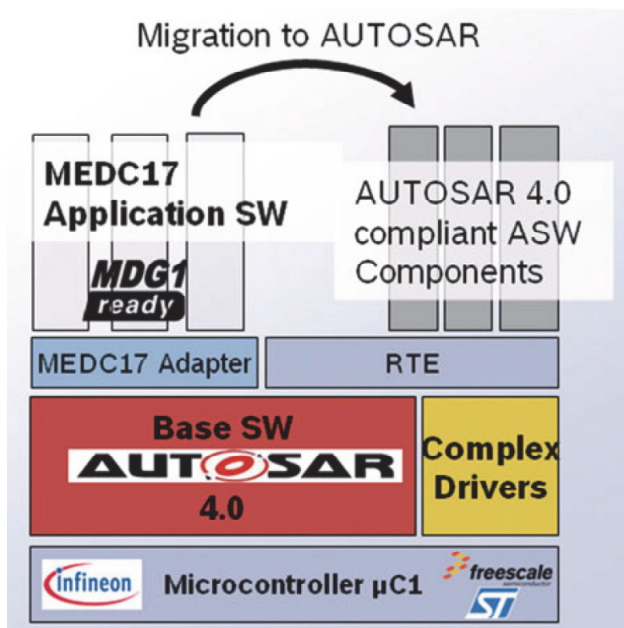


Figure 3: Software architecture of MDG1 ECU, (Rüger et al. 2013)

3.1 Considerations for the prototypical porting of FMUs to the ECU

The prototype is intended to work with source code FMUs from publicly available modeling tools. These tools were selected by inspection of the generated source C-code, w.r.t. the criteria listed in Section 4.2. As even the most suitable C-code did not fulfill all our requirements, modifications of the C-code are necessary. For this purpose a conversion tool was developed to perform this semi-automatically.

When considering the usage on FMI for embedded software, one has to choose between model exchange or co-simulation FMUs. Both types of FMUs have their pros and cons: co-simulation is closer to task-based execution in ECU software, while model exchange can interface special re-usable numerical algorithms optimized for the ECU that can be provided within a solver library.

The choice of the FMU type also depends on what the modeling tool can export: Some tools export very good variable step solvers for co-simulation FMUs intended for the PC-world but with no chance to compile and use such solvers for an embedded target. More suited for embedded targets are inline integration methods (Elmqvist et al. 1995).

On the other hand, model-exchange FMUs can take advantage of optimized solvers and numerics for the target platform without the above mentioned target dependency of the FMU. For this reason both, co-simulation and model exchange FMUs, are selected. However, regarding real-time capability, especially for model exchange FMUs, the prototype only supports a subset of the FMI standard necessary for the description of continuous ODEs neglecting events. The assumptions and limitations of the implementation are described in Section 3.4.

3.2 A prototypical workflow to bring FMUs on Bosch ECUs

This prototypical workflow addresses the transformation that an FMU must go through to be a part of an ECU software component, by utilizing the available ECU resources and numerical capability. The whole conversion process can be done in a preparation phase offline on a development platform (typically a PC), where

- the FMU is unzipped,
- the `modelDescription.xml` file is parsed and mapped to the corresponding configuration XML file on the ECU software side, and
- the C-code is manipulated so that it can be compiled for the target ECU within the standard ECU software architecture before finally
- the manipulated C-code is included in an application software component that can be included in the overall ECU software.

Then, the software can be compiled and flashed to the ECU and then executed and validated.

3.2.1 Mapping of configuration files and interface definitions

The data description file of the FMI standard `modelDescription.xml` must be converted to the data description format for ECUs (e.g., ARXML in AUTOSAR). This is performed analogously to the way described by (Thiele et al. 2011). This task is done by the first stage of a script-based conversion tool (see Figure 4).

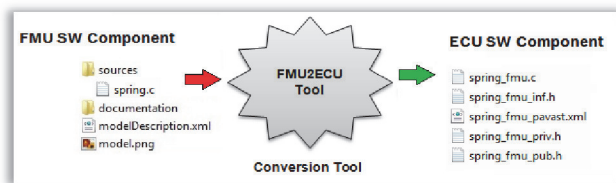


Figure 4: FMU file artifacts converted automatically into an ECU compatible software component

3.2.2 Conversion of C-source code from FMUs to ECU software C-code

Some changes like the selection of suitable data types (e.g., single precision float for `fmi2Real`) can be configured in the platform dependent configuration files `fmi2TypesPlatform.h` and `fmi2Functions.h`.

For other changes the second stage of the tool-supported conversion process applies: the C-code contained in the source code FMUs is converted to C-code that can be compiled for an ECU with the standard toolchain for Bosch engine control software. Since the FMU source code is generated from simulation tools typically operating in a PC environment, there are some code adaptations involved in the process of running them on ECU. However, the goal is to keep the code change as low as possible. Some examples of the changes of the C-code are:

- All macro definitions in `fmuc.c` must be moved to private headers to avoid multiple definitions while compiling multiple FMUs together. For example, definitions such as the following should be put in private headers:

```
#define NUMBER_OF_REALS XX
#define NUMBER_OF_INTEGERS YY
```
- Information from FMUs that must be exposed to the outside shall be defined in public header files, e.g., reference to the FMU interfaces or function declaration prototypes such as

```
void modelName_fmi2instantiate(...);
```
- Some header inclusions generated in FMU source files like `stdio.h`, `math.h` etc., must be removed or excluded since they are not available on the embedded platform.

- Mathematical function calls must be mapped to the available functions from the AUTOSAR base software.
- Floating point variables and constants within the code have to be converted to single precision. A notable compiler specific change is that for any arithmetic expression in model equations where float constants are used, they must be suffixed by `f` to enable the compiler to identify that it is a single precision float variable and not double precision. This means that the original code `mu * ((1.0 - x0 * x0) * x1) - x0;` is replaced by `mu * ((1.0f - x0 * x0) * x1) - x0;`
- Implicit type castings are replaced by explicit type castings.
- Any explicit print functions like `fprintf()`, `printf()` should be removed.

The structure of the C-code contained in source code FMUs is very specific to the exporting tool. At the moment FMUs generated from three different tools are supported. The conversion of the C-code is performed mainly automatically, but still the inspection of the modified C-code and some manual adaptations are necessary.

3.2.3 Modification of memory allocation

On the ECU, dynamic memory allocation is not allowed. Instead, the memory demand must be known at compile time. Therefore, the required memory must be pre-allocated and later assigned via the FMI callback function `allocateMemory()`. However, it is not sufficient for the ECU implementation of the callback function to return just a pointer to any free space in memory. Instead, memory for each variable is pre-allocated individually and the correct memory location for each allocation request must be returned. This is important for debugging and calibration. The respective tools must be able to map the variable names to the resp. space in memory. For instance, a model might use a `struct` containing the parameters *a*, *b*, and *c*. Here, the calibration tool must be able to unambiguously map the variables *a*, *b*, and *c* to their respective locations in memory.

3.2.4 Concept of FMU computation algorithm

The modified C-code is called by the “FMU computation algorithm” which takes the role of the solver in the case of model exchange FMUs and the master algorithm in the case of co-simulation FMUs. This FMU computation algorithm is the application software component that can communicate with the other parts of the ECU software and is scheduled by the operating system.

The FMU computation algorithm can make use of solver libraries in the case of model exchange. It

interacts with the modified C-code from the FMUs via the standard FMI interface functions according to the supported part of the FMI standard calling sequence.

This design enables easy integration of FMU components as ECU components, thereby enabling interfacing with the FMU computation algorithm, solver libraries, and other ECU components. A graphical representation of the FMU software components' organization inside the ECU is shown in Figure 5.

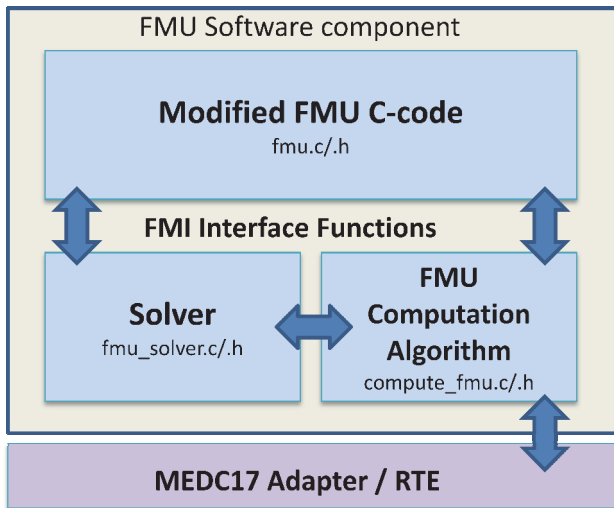


Figure 5: FMU software component for model exchange FMUs

As shown in the depicted setup, the FMU is accessed via the FMI interface functions, thus keeping the FMU's C-code with minimal changes while executing it on the ECU. The FMU computation algorithm is a combined set of functions defined, organized and distributed across different source files under a common container.

3.2.5 Example: double mass spring damper model

Several FMUs have been ported to the Bosch MDG1 ECU with different properties of the physical models such as number of states (up to >10), stiffness of the included ODEs and physical domains covered. The prototype workflow allows bringing physical models very easily on the ECU in a fraction of the time necessary to hand-code the model and solvers from scratch.

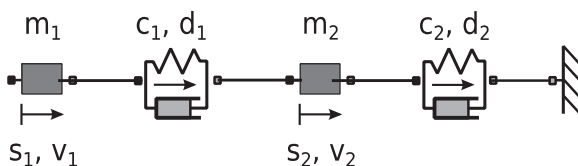


Figure 6: Schematic representation of the double mass spring damper model

To demonstrate the execution of FMUs running on an ECU, a stiff model of a double mass spring damper as depicted in Figure 6 is considered. This example serves as a benchmark problem for a stiff powertrain model that could be used in applications listed in Section 2.3. For this simple model, source code FMUs were generated with several Modelica-based simulation tools. The size of the contained C-code differed from 54kByte to 2.9MByte, i.e., differing by a factor of more than 50. This correlated also with the complexity and "readability" of the contained C-code and is a first indication, how feasible it is to re-use this C-code on embedded targets.

For the above model, the system of equations is given by

$$\begin{aligned} \dot{s}_1 &= v_1 \\ m_1 \dot{v}_1 &= c_1(s_2 - s_1) + d_1(v_2 - v_1) \\ \dot{s}_2 &= v_2 \\ m_2 \dot{v}_2 &= c_1(s_1 - s_2) + d_1(v_1 - v_2) - c_2 s_2 - d_2 v_2 \end{aligned} \quad (1)$$

with the following parameters:

$$\begin{aligned} m_1 &= m_2 = 1 \text{ kg}, \quad c_1 = c_2 = 1 \text{ N/m}, \quad d_1 = 100 \text{ Ns/m}, \quad d_2 = 1 \text{ Ns/m} \\ \text{and initial conditions:} \\ s_1 &= 2 \text{ m}, \quad v_1 = 0 \text{ m/s}, \quad s_2 = 2 \text{ m}, \quad v_2 = 0 \text{ m/s.} \end{aligned}$$

The solution calculated with a linear-implicit Euler solver with a step size of 10 ms demonstrates the expected damped oscillation of the masses. An explicit Euler solver is unstable for the same step size, as shown in Figure 7.

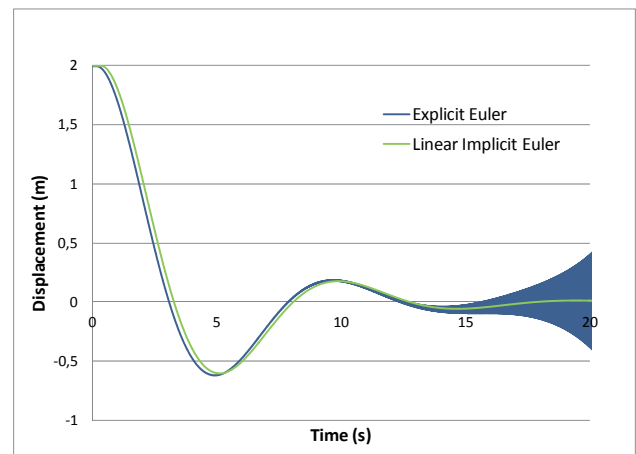


Figure 7: Displacement of mass m_2 provided by explicit and linear implicit Euler methods

This well known behavior of explicit solvers for stiff ODEs with large time steps also demonstrates the benefit of FMI for porting physical models on the ECU: (linear-)implicit solvers can be easily realized either by:

- Co-simulation FMUs generated by exporting tools supporting inline integration for implicit discretization methods of ODEs (Elmqvist et al. 1995)
- Model-exchange FMUs providing a standardized interface between the model equations (even with the possibility to calculate the Jacobian matrix in FMI 2.0) and an optimized implementation of numerical algorithms for the target platform.

In this example, a model exchange FMU generated from a commercial tool is used as well as our own implementation of explicit and linear-implicit Euler solvers on the Bosch MDG1 ECU.

3.3 Validation of FMUs running on the ECU

The computation results of the FMUs ported to the MDG1 ECU are validated in several steps:

- Creating reference signals by simulating the FMU offline on a PC. The FMU can also be embedded in a closed loop system model to derive meaningful stimuli for the further tests.
- Compiling the modified FMU within an ECU software with the Bosch standard toolchain and creating a Windows DLL that can be tested within the ETAS test tool RT2.
- Compiling the modified FMU within an ECU software with the Bosch standard toolchain for the target processor and running it on virtual hardware (Leupers et al. 2012).
- Running the FMU on the real target platform (Bosch ECU test board or series product). Stimulate and measure with an ETAS LABCAR Hardware-in-the-Loop (HiL) setup or a measurement and calibration tool such as ETAS INCA.

3.4 Restrictions of the prototype implementation

There are also some restrictions in the prototypical implementation of FMU like no event handling for model exchange and no variable step size for solvers. Otherwise one would have to take additional measures to guarantee real-time behavior and to map calculations to ECU tasks. Special assumptions on the contained C-code are made and adapted to the prototype to cope with FMUs generated from three different tools (including Modelica-based tools) which address the desire of using FMUs from commercial tools. However, C-code from many commercial tools is not suitable to run on an ECU due to their size and complexity since it was not intended to run on an embedded system.

At the moment, the prototype does not yet fulfill all requirements, e.g., regarding compliance with the software development guidelines for the C programming language by the Motor Industry Software

Reliability Association (MISRA 2013), for series engine control software.

With the successful prototypical implementation, the next steps will be:

- Usage of FMUs for the computation of physical models as virtual sensors or in advanced control and diagnosis tasks as described in Section 2.3.
- Extension of the offline preprocessing and calculation algorithm for connected FMUs as well as the connection to other software components
- Full AUTOSAR support

4 Key findings and outlook

Based on the sample FMUs implemented on a Bosch ECU, the key findings are summarized and an outlook is given.

4.1 Aspects for embedded systems already contained in the FMI standard

In the original development of the FMI, some consideration was already given to the usage of FMI on embedded systems (cf. the FMI 2.0 standard), resulting in the following features which are required by such systems:

- Source code FMUs come with C-code, the most common language to program embedded systems.
- It is easy to replace data types (double precision float to single precision float).
- The interface description in an XML file can be mapped easily to corresponding XML files for ECU software components.
- Co-simulation FMUs with fixed communication step size can be mapped easily to ECU tasks with periodic activation
- Callback functions are defined for memory allocation and logging, which can be replaced or disabled by special mechanisms for the target platform.

4.2 Key findings: changes to source code FMUs for embedded targets

On ECUs, one is confronted with computation limitations compared to the offline world or the real-time PC environment. These limitations should be reflected in an FMU that is suitable for applications running on an ECU.

- **Limitation 1: Memory**
The FMI purposefully leaves the organization of a model's data (e.g., parameters, internal variables) to the FMU in order to achieve maximum freedom. In contrast, ECU software is organized with respect to memory to allow transparency, simplicity, and efficiency. That means, if data structures are left to the freedom of the implementer, they still have to be transparent to the outside at least so far as to allow

parameterization and signal observation. Currently, this is not possible with the FMI since parameters, states, inputs, and outputs are not exposed directly to the outside but hidden behind access functions.

- **Limitation 2: Event handling**
In general, events could increase the runtime for real-time systems in an unpredictable manner. There may be any number of events within a second which all trigger their respective event handling algorithms. Thus, in order to guarantee that models with event handling satisfy real-time requirements one will have to take extra measures.
- **Limitation 3: Algebraic loops**
Similar to event handling, algebraic loops generally have an unpredictable impact on the run time of an FMU where they require an iterative solution. This restriction applies both to connected FMUs and to iterative solution methods within one FMU.
- **Limitation 4: User-interaction is impossible**
Several features which make sense for offline simulations are either overhead or even dangerous for computations on the ECU. Such features which are either supported or not explicitly forbidden in the FMI include logging and I/O operations such as `print()`.
- **Limitation 5: Support functions**
Support functions such as available from the `math.h` library are usually not available on the ECU. State of the art ECUs provide their own set of support functions through an AUTOSAR library. However, function names and usage of these are generally different from their offline counterparts.
- **Limitation 6: C-code compliance**
Strict coding guidelines apply for source code that is executed on an ECU. Such requirements are standardized by the Motor Industry Software Reliability Association (MISRA 2013).
- **Limitation 7: Cross-compilation and object code FMUs**
As ECUs are quite special platforms, they require special compilers and build processes as well. Object file FMUs have to be cross-compiled accordingly which requires the availability of the suitable toolchain for the target platform
- **Limitation 8: Available data types**
In order to provide optimized code, the set of available data types should be enlarged. For example, one should be able to distinguish a `uint8` from a `uint32` variable.

Beyond the mentioned restrictions, to achieve a convincing performance of FMUs running on an ECU, careful consideration of the target platform is required:

- The solver must be able to make use of the platform's computing capabilities, which differ widely from one platform to another.
- Computations are mostly performed using single precision floats.
- Heap and stack usage must be minimized.

4.3 Outlook towards a future FMI variant for embedded systems

The limitations listed in the previous section will have to be addressed by a possible future FMI standard. Major changes to the existing FMI 2.0 standard will be necessary that it does not seem to be possible to include them very easily, as there will be not only requirements on the interface such as for FMI today, but also on the contained C-code. Thus, a variant of the FMI standard especially for automotive embedded targets and a connection to the AUTOSAR standard is desirable. FMUs generated according to such a standard could then be executed on ECUs without the conversion steps presented in Section 3 of this paper. Such FMUs could really be seamlessly used for all cycles of the development process until running on automotive embedded systems.

5 Summary

In this paper, it is shown by a prototype that the current FMI standard with some modifications which are highly tool-dependent allows the computation of FMUs of physical models on an ECU (as a representative for embedded controllers) as long as the FMU satisfies some assumptions and limitations. This was demonstrated for FMUs generated from different tools on a Bosch MDG1 engine control unit, where the modified FMUs have been included in the real software build process. The prototype can be used for rapid control prototyping with physical models.

For usage of FMUs out of the box in productive ECU software, the standard will have to be modified, mostly enforcing the above mentioned restrictions (Section 4.2). ECU software must be lean, efficient, and above all, safe. We foresee the benefits for the establishment of an "FMI for automotive embedded systems" for seamless model-based design until the execution on the target platform.

Acknowledgements

We would like to thank Martin Otter, Andreas Pfeiffer, and Jonathan Brembeck from DLR as well as our colleagues Andrea Flexeder, Eckart Mayer-John, Matthews Peter, Dibakar Mahalanabish, Naresh Mandipalli, Wolfgang Lengerer, and Marcus Bossler for fruitful discussions.

References

- AUTOSAR Consortium, Autosar Standard 4.2, available at <http://www.autosar.org/>, 2015
- Bertsch, C., Ahle, E., Schulmeister, U., The Functional Mockup Interface - seen from an industrial perspective, In: Proceedings of the 10th International Modelica Conference 2014, Lund, Sweden
- 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, In: Proceedings of the 8th International Modelica Conference 2011, Dresden, Germany
- Blochwitz, T., Otter, M., Akesson, J., Arnold, M., Clauß, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A., The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models, In: Proceedings of the 9th Modelica Conference 2012, Munich, Germany
- Brembeck, J., Pfeiffer, A., Fleps-Dezasse, M., Otter, M., Wernersson, K., Elmqvist, H., Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface, In: Proceedings of the 10th International Modelica Conference 2014, Lund, Sweden
- Chombard, P., Multidisciplinary modeling and simulation speeds development of automotive systems and software, ITEA2 innovation report, 2012, published online: <https://itea3.org/project/modelisar.html>
- Ding, S. X., Model-Based Fault Diagnosis Techniques, Springer, 2nd edition, London 2013
- Elmqvist, H., Otter M., Cellier, F.E.: Inline Integration: A new mixed symbolic/numeric approach for solving differential-algebraic equation systems, In: Proceedings of ESM'95, European Simulation Multiconference, 1995
- Franke, R., Mathematical optimization of dynamic systems with OpenModelica, Annual OpenModelica Workshop 2015, published online: https://openmodelica.org/images/docs/openmodelica2015/OpenModelica2015-talk02-Franke_Optimization.pdf
- Leupers, R., Martin, G., Plyaskin, R., Herkersdorf, A., Schirrmeister, F., Kogel, T., Vaupel, M., Virtual Platforms: Breaking New Grounds, IEEE DATE Conference, Dresden 2012
- Mitrohin, C., FMI in LABCAR HiL; From MiL to SiL towards HiL, FMI-Tutorial, 10th International Modelica Conference 2014, Lund, Sweden
- MISRA Consortium, MISRA C: 2012 Guidelines for the use of the C language in critical systems, 2013, available from <http://www.misra.org.uk>
- Rüger, J.-J., Wernet, A., Kececi, H.-F., Thiel, T., MDG1: The New, Scalable, and Powerful ECU Platform from Bosch, Proceedings of the FISITA 2012 World Automotive Congress - Volume 6: Vehicle Electronics, Springer, 2014
- Seuling, S., Hamedovic, H., Fischer, W., and Schuerg, F., Model Based Engine Speed Evaluation for Single-Cylinder Engine Control, SAE Technical Paper 2012-32-0044, 2012
- Thiele, B.; Henriksson, D., Using the Functional Mockup Interface as an Intermediate Format in AUTOSAR Software Component Development, In: Proceedings of the 8th International Modelica Conference 2011, Dresden, Germany
- Wagner, A., Bleile, T, Lux, S., Fleck, C., Method for real time capability simulation of an air system model of an internal combustion engine, US Patent US 8321172 B2, 2008

Methodology for Obtaining Linear State Space Building Energy Simulation Models

Damien Picard¹ Filip Jorissen^{1,2} Lieve Helsen^{1,2}

¹Mechanical engineering, KU Leuven, Belgium, {damien.picard, filip.jorissen}@kuleuven.be

²EnergyVille, Waterschei, Belgium, lieve.helsen@energyville.be

Abstract

Optimal climate control for building systems is facilitated by linear, low-order models of the building structure and of its Heating, Ventilation and Air Conditioning (HVAC) systems. However, obtaining these models in a practical form is often difficult, which greatly hampers the commercial implementation of model predictive controllers. This work describes a methodology for obtaining a linear State Space Model (SSM) of Building Energy Simulation (BES) models, consisting of walls, windows, floors and the zone air. The methodology uses the Modelica library IDEAS to develop a BES model, including its non-linearities, and automates its linearisation. The Dymola function `linearize2` is used to generate the state space formulation, facilitating further mathematical manipulations, or simulation in different environments. Optionally this model can then be reduced for control purposes using model order reduction (MOR) techniques. The methodology is illustrated for the zone air temperature in an office building. For this case, the absolute error between the non-linear BES and its SSM remains under 1 K and its yearly average is 0.21 K. The original 50 states SSM could furthermore be reduced to 16 states without significant loss of accuracy.

Keywords: model predictive control, Dymola, building energy simulation, linearisation, model order reduction.

1 Introduction

Building climate control uses around 18% of the total end energy in Europe (Perez-Lombard et al., 2008). One way of reducing energy use is to develop more efficient control algorithms for the production and distribution of heat and cold in buildings. Recent research has shown that (near) optimal controllers such as Model Predictive Control (MPC) can greatly improve the energy efficiency of buildings compared to traditional rule-based-controllers (Gyalistras and Gwerder, 2009; Verhelst, 2012). However, its practical implementation is hampered due to the difficulty of finding a controller model that is simple enough to allow optimization within

a reasonable computation time but still accurate enough to correctly predict the building behaviour. Linear models are preferred since efficient optimization algorithms can then be used (Kummert, 2001; Sturzenegger et al., 2012).

Controller models are often obtained using system identification, i.e. fitting reduced order models based on measurement data. Obtaining controller models for buildings is an active research topic due to the complexity of the systems and due to the difficulty or even impossibility of performing experiments allowing the identification of multi-input, multi-output building models (Sturzenegger et al., 2014). An alternative approach is to create models based on physical insight and knowledge about the system. Lehmann et al. (2013) showed that building energy simulation (BES) models are only weakly non-linear. They set up a relative complex linear model based uniquely on physical data, which was able to mimic the non-linear TRNSYS BES model with an error smaller than 1 K. The accuracy of the model is not enough for design purposes but it is sufficient for MPC or sensitivity analysis. Sturzenegger et al. (2014) automated their approach for deriving state space models for MPC applications using the BRCM Matlab toolbox. The toolbox needs a considerable amount of information such as an EnergyPlus input file.

In this work, we propose an automated way of obtaining accurate linear BES models based on a non-linear model implementation in Dymola using the IDEAS library (Baetens et al., 2015). Section 2 describes the non-linearities of BES models together with common simplifications and Section 3 explains the linearisation technique. Section 4 describes the linearisation methodology in IDEAS and Section 5 shows a validation of the methodology. Section 6 briefly discusses a model order reduction technique for the linear model and their use for optimal controllers. Main conclusions are summarized in Section 7.

2 Non-linearities in Building Energy Simulation Models and Common Simplifications

Typically, BES models contain three major sources of non-linearities. The first is longwave radiation, which is typically described using the Stefan-Boltzmann law. The second is the absorption of incident solar radiation by windows, which is a function of the incidence angle. The third is convective heat transfer, which is usually described using correlations for the convective heat transfer coefficient. These non-linear equations are first described in this section, then a linearisation technique is proposed. Other non-linearities in real buildings exist (e.g temperature dependent emissivity, pressure dependent air leakage, ...) but they are rarely modelled. They will not be treated in this work.

Radiation Radiation is described by the non-linear Stefan-Boltzmann law which is given by Eq. 1 for two grey-bodies with surface areas A_1 and A_2 .

$$\dot{Q}_{1 \rightarrow 2}(t) = \sigma F_{1 \rightarrow 2} A_1 (T_1^4(t) - T_2^4(t)) \quad (1)$$

$\dot{Q}_{1 \rightarrow 2}$ and $F_{1 \rightarrow 2}$ are the heat transferred from surface 1 to 2 and their view factor respectively, $\sigma = 5.670373 \times 10^{-8} \text{ W}/(\text{m}^2 \cdot \text{K}^4)$ the Stefan-Boltzmann constant, and T_i the temperature of body i .

Radiative heat transfer between room surfaces is often approximated using the Mean Radiant Temperature model (e.g. in TRNSYS TYPE 56 (S.A. Klein et al., 2010)) or using the Radiant Star Temperature model (e.g. in IDEAS (Baetens et al., 2015)) since it greatly simplifies the computations without a significant loss in accuracy (Liesen and Pedersen, 1997). This radiant star temperature T_{star} is derived from the energy conservation equation in the radiant node and the temperature of each surface A_k is calculated using a distribution coefficient R_k :

$$\dot{Q}_{k \rightarrow \text{star}}(t) = \frac{\sigma A_k}{R_k} (T_k^4(t) - T_{\text{star}}^4(t)) \quad (2)$$

Eq. 2 is often linearised around nominal temperatures $T_{k,\text{nom}}$ and $T_{\text{star},\text{nom}}$ (Eq. 3), which is an accurate approximation for small temperature differences. Figure 1 shows the approximation error for the heat exchange between two black bodies with view factor equal to one.

$$\dot{Q}_{k \rightarrow \text{star}}(t) \simeq c (T_k(t) - T_{\text{star}}(t)) \quad (3)$$

$$c = \frac{\sigma A_k}{R_k} ((T_{k,\text{nom}} + T_{\text{star},\text{nom}})(T_{k,\text{nom}}^2 + T_{\text{star},\text{nom}}^2)) \quad (4)$$

The longwave radiation heat flow $\dot{Q}_{lw,k}(t)$ between exterior surface k of the building with longwave emissivity

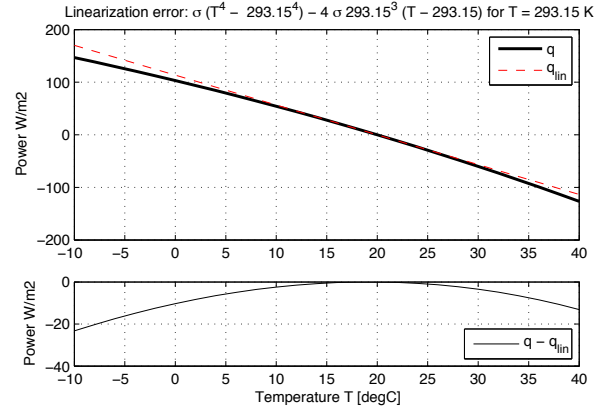


Figure 1. Error made by the linearisation of the radiative heat transfer equation between two black bodies with view factor one.

$\varepsilon_{lw,k}$ and its environment can be modelled as:

$$\begin{aligned} \dot{Q}_{lw,k}(t) = & \sigma \varepsilon_{lw,k} A_k (T_{s,k}^4(t) - F_{ce,k} T_{ce}^4(t) \\ & - (1 - F_{ce,k}) T_{db}^4(t)) \quad (5) \\ F_{ce,k} = & \frac{1 + \cos i_k}{2} \end{aligned}$$

with $T_{s,k}(t)$, $T_{ce}(t)$, $T_{db}(t)$ the surface, celestial dome and dry bulb temperature respectively, $F_{ce,k}$ the view factor between the surface k and the celestial dome, and i_k the inclination of the surface. This equation is linearised by default in IDEAS as:

$$\dot{Q}_{lw,k}(t) \simeq c (T_{s,k}(t) - \sqrt[4]{F_{ce,k} T_{ce}^4(t) + (1 - F_{ce,k}) T_{db}^4(t)}) \quad (6)$$

with c a parameter defined similar to Eq. 4.

Finally, the shortwave solar irradiation absorbed by exterior surface k equals:

$$\dot{Q}_{sw,k}(t) = \varepsilon_{sw,k} A_k E_{e,k}(t) \quad (7)$$

with $E_{e,k}(t)$ the incident solar irradiation on surface A_k as a function of time.

Absorption and transmission through glazing Heat absorbed or transferred through windows is typically highly non-linear as it depends on the spectral properties of the window, on the angle of incidence of the sun and on possible shading. Typically, the window properties are pre-computed using specialized software and delivered as an input to the simulation software. IDEAS uses the software `Window 4.0` (Finlayson et al., 1993) to pre-compute window spectral properties but it computes the amount of absorbed and transmitted light during the simulation, requiring trigonometrical transformations and lookup tables, which are non-linear functions.

Convective heat transfer Two types of convective heat transfer are present in buildings: exterior, forced convection by the wind, and interior, natural convection when forced ventilation is absent.

In IDEAS, the external convective heat transfer rate $\dot{Q}_{cv}(t)$ between an exterior surface with area A_k and the outdoor air is based on Defraeye et al. (2011):

$$\begin{aligned} \dot{Q}_{cv,k}(t) &= h_{cv}(t)A_k (T_{db}(t) - T_{s,k}(t)) \\ h_{cv}(t) &= \max \left\{ 5.01(v_{10}(t))^{0.85}, 5.6 \right\} \text{ W/m}^2\text{K} \end{aligned} \quad (8)$$

with convective heat transfer coefficient $h_{cv}(t)$, dry bulb ambient temperature $T_{db}(t)$, surface temperature $T_{s,k}(t)$ and the undisturbed wind speed at 10 meters above the ground $v_{10}(t)$.

Eq. 8 is non-linear even if the convection coefficient is an input due to the multiplication of input with input ($h_{cv}(t)T_{db}(t)$) and inputs with state ($h_{cv}(t)T_{s,k}(t)$). If the nominal values of $T_{db}(t)$ and $T_{s,k}(t)$ are equal, Eq. 8 can be linearised as:

$$\dot{Q}_{cv,k}(t) \simeq \bar{h}_{cv}A_k (T_{db}(t) - T_{s,k}(t)) \quad (9)$$

with \bar{h}_{cv} the yearly average of the exterior convection coefficient.

The interior convective heat transfer rate of a wall, ceiling or floor with surface area A_k and an air node is computed as:

$$\begin{aligned} \dot{Q}_{cv,k}(t) &= h_{cv,k}(t)A_k (T_{db}(t) - T_{s,k}(t)) \\ h_{cv,k}(t) &= n_{1,k} D_k^{n_{2,k}} |T_{db}(t) - T_{s,k}(t)|^{n_{3,k}} \end{aligned} \quad (10)$$

with D_k the hydraulic diameter, and coefficients $n_{i,k}$. The value of the coefficients are $n_{1:3} = \{1.823, -0.121, 0.293\}$ for vertical surfaces, $n_{1:3} = \{2.175, -0.076, 0.308\}$ for heated floors and cooled ceilings and $n_{1:3} = \{0.704, -0.601, 0.133\}$ for cooled floors and heated ceilings (Awbi and Hatton, 1999).

These interior convection equations can be linearised in IDEAS using an average value for h_{cv} :

$$h_{cv,k} \simeq n_{1,k} D_k^{n_{2,k}} |\Delta T_{nom}|^{n_{3,k}} \quad (11)$$

with ΔT_{nom} the nominal temperature difference.

Heat diffusion through walls and floors Heat transfer through walls and floors is characterized by convective and radiative heat transfer at the surfaces and conduction through the solid layers. The latter is governed by a partial differential equation (PDE). It extends in three spatial dimensions and in time. However, the heat transfer through walls and floor can often be approximated using a one dimensional PDE due to the low thickness to height and width ratio. The equations can then either be solved using discrete Laplace transform (e.g. TRNSYS) or using a finite volume method (e.g. EnergyPlus (Strand et al., 1999)). In IDEAS, the finite volume method is used, leading to a set of linear equations.

3 Linearisation Technique

The linearisation of a function consists of the first order term of the Taylor expansion of this function around a working point. Given a deterministic non-linear dynamic system:

$$\begin{aligned} \dot{x} &= f(x, u) \\ y &= g(x, u) \end{aligned} \quad (12)$$

where $x \in \mathbb{R}^{n_x}$ are the states, \dot{x} are their derivatives, $u \in \mathbb{R}^{n_u}$ the inputs, and $y \in \mathbb{R}^{n_y}$ the outputs. The linearisation of Eq. 12 around point $p_* \triangleq (x_*, u_*)$ is defined as:

$$\begin{aligned} \dot{x} &= f(p_*) + \left. \frac{\partial f}{\partial x} \right|_{p_*} (x - x_*) + \left. \frac{\partial f}{\partial u} \right|_{p_*} (u - u_*) \\ &\triangleq f(p_*) + \mathbf{A}\tilde{x} + \mathbf{B}\tilde{u} \\ y &= g(p_*) + \left. \frac{\partial g}{\partial x} \right|_{p_*} (x - x_*) + \left. \frac{\partial g}{\partial u} \right|_{p_*} (u - u_*) \\ &\triangleq g(p_*) + \mathbf{C}\tilde{x} + \mathbf{D}\tilde{u} \end{aligned} \quad (13)$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are constant matrices.

The Dymola built-in function `linearize2` of the Modelica Linear System2 library provides the possibility of linearising Modelica models (Otter, 2014). The hybrid differential-algebraic equation system is treated as an ordinary differential equation system at the linearisation point and the partial derivatives of the functions f and g are obtained by evaluation of the analytical Jacobian if it is available. Otherwise a central difference method is used. The function can also be used to transform a linear model into a SSM.

It should be noted that even for a linear system, the linearisation point p_* used by the function `linearize2` should be chosen carefully to avoid numerical noise. The states x_* can be set using initial equations or `start` values. The inputs u_* can be set using `start` values. The default `start` value for the inputs in Dymola is zero which can lead to significant error when evaluating the derivatives using the central difference method.

4 Linearisation Methodology in IDEAS

This section describes how IDEAS was adapted to automatically obtain a state space formulation of a BES model in Dymola. Firstly the linearization of the equations is discussed, followed by the model structure requirements for SSM's. Finally the SSM structure is described.

4.1 Linearisation of the equations

Here we describe how the non-linear equations of the Modelica BES models are conditionally linearised or

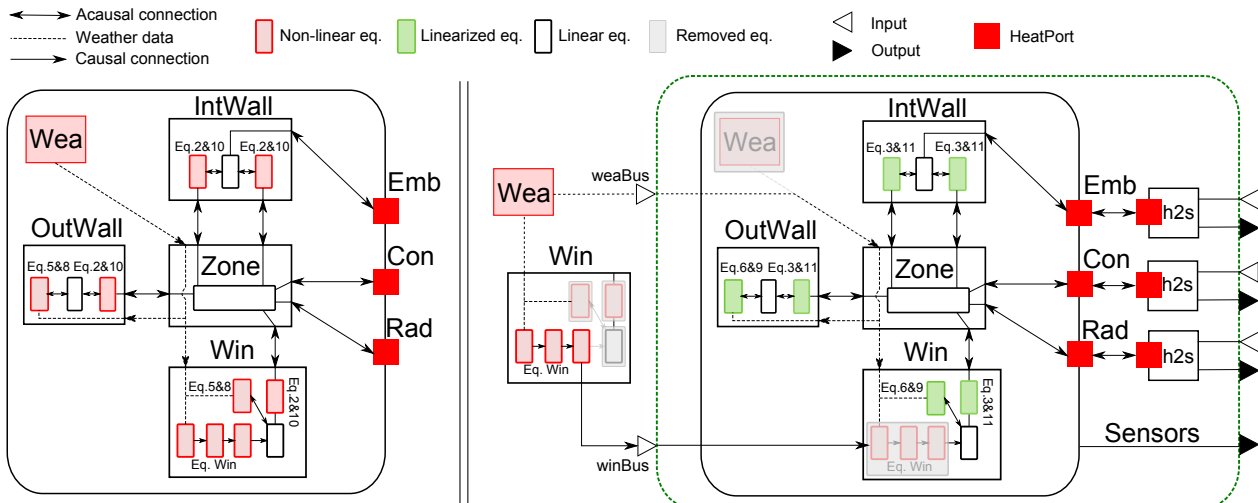


Figure 2. Left: original model with non-linear equations. Right: Adjusted model structure with moved and/or linearized non-linear equations. Component models are outer wall ‘OutWall’, interior wall ‘IntWall’, Window ‘Win’, weather model inputs ‘Wea’ and HeatPorts embedded (Emb), convective (Con) and radiative (Rad). White triangles represent inputs to the model, whereas black triangles represent outputs of the model.

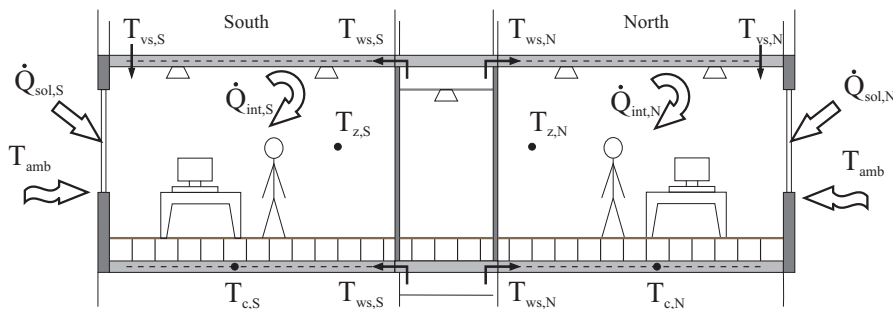


Figure 3. Illustration of the office building section, (Sourbron et al. (2013), p 5)

moved outside the model and replaced by model inputs. Note that the moved equations should not depend on any state variables.

Radiation As described in the previous section, all longwave radiation equations can be linearised accurately. If `linearise = true`, Eq. 2 and Eq. 5 are replaced by Eq. 3 and Eq. 6, respectively, where the square root term is transformed into a model input for each different orientations and inclination. The solar irradiation $E_e^{(k)}(t)$ required for the shortwave absorption is also converted into a model input per orientation and inclination.

Window models Window models contain equations for calculating the solar irradiance, the impact of shading and the amount of heat that is absorbed and transmitted through the window. These are non-linear equations indicated in Figure 2 by ‘Eq. Win’. Linearising these equations would introduce large errors. Linearising them at may for instance have the consequence that the solar position and corresponding incidence angles become fixed, which can cause a large underestimation of the so-

lar gains for windows. Therefore the absorbed and transmitted heat flow rates are calculated outside of the model and they are inputs to the linearised model, as indicated in the right of Figure 2. Each window model is instantiated twice, once inside and once outside of the linearised model. The grey boxes in Figure 2 indicate which equations are removed and replaced by inputs. Note that the window model is thereby split into two parts. A bus connector `winBus` for each of the n_{win} windows is used for connecting the inputs.

Convective heat transfer The interior convective heat transfer is linearised using Eq. 11. ΔT_{nom} was chosen equal to the mean absolute temperature difference between the window or wall and the zone air temperature. The exterior convective heat transfer coefficient is simplified by using the yearly average convective heat transfer coefficient \bar{h}_{cv} . These linearisations are indicated on Figure 2 using green rectangles.

The remaining model equations, like thermal conduction equations, are already linear and they are retained.

4.2 State space formulation

In the previous paragraphs all non-linear equations are removed from the building envelope model. This linear model needs to be converted into state space format. This requires that all exterior connections are either inputs or outputs, otherwise Dymola does not detect the connections. However, `HeatPort` connections `Emb`, `Con` and `Rad` contain variables `T` and `Q_flow` that do not specify whether they are inputs or outputs. Each `HeatPort` for the room thermal gains is therefore connected to an input-output block `h2s`, which either sets heat flow rate `Q_flow` to a fixed input and temperature `T` to an output or the other way around.

In order to propagate weather data inputs to all sub-models, one weather bus `weaBus` with prefix `input` is connected to each zone. The zone further propagates this data to all its connected surfaces (walls, windows, ...) as indicated by the dotted lines in Figure 2.

4.3 State space model structure

All non-linear equations are now removed and all connections are either defined as an input or as an output. The state space formulation can now be obtained by using the `linearize2` function on the model containing all components of the dashed green box in Figure 2. This function returns matrices **A**, **B**, **C** and **D**. The SSM inputs u are the heat flow rate or temperature for thermal gains of the zones, the weather bus and n_{win} window buses. Outputs are either the temperature or the heat flow rate of the transformed `HeatPorts`. Additional outputs can be defined in the linearised model by adding `RealOutput` components.

5 Validation

In this section, the methodology is applied to a test case. The case is firstly described after which the methodology is validated.

Case description The validation uses the cut-out of a typical office building with South and North oriented facades described by Sourbron et al. (2013) (See Figure 3). We only consider the building structure, which consists of three zones (a corridor, a south-oriented and a north-oriented zone) each equipped with a thermally activated ceiling and floor composed of multiple layers (floor tiles, air layer, screed, and reinforced concrete), two external walls composed of multiple layers (plaster, concrete blocks, mineral wool, and bricks), and two windows. Each zone has a convective and a radiative heat gain input and heat can also be injected at the core of the thermally activated building parts.

The model is implemented with all details above in Modelica using the IDEAS library (Baetens et al., 2015).

Table 1. Comparison between three models based on equation types and equation formats.

	Ref	Lin	SSM
Convection	non-linear	linear	linear
Radiation	non-linear	linear	linear
Model inputs	non-linear	non-linear	non-linear
Other equations	linear	linear	linear
SSM formulation	no	no	yes

The model has 8434 variables and 50 differentiated states. Once linearised, the model has 52 inputs. The model uses the weather data of Uccle (Belgium).

Each of the heat flow rate inputs is set equal to the sum of the two sinusoids of Equations 14-15, with $t = 0$ at new year. The sinusoid with a period of one day and one year respectively represent internal gains, and heating or cooling delivered by the HVAC system. The sinusoid parameters are tuned such that the zone temperature remains around 22 °C.

$$\sin_1 = 4 + 4 \sin\left(\frac{2\pi t}{86400} - \frac{\pi}{2}\right) \quad (14)$$

$$\sin_2 = 13 \sin\left(\frac{2\pi t}{31536000} - 1.4\right) \quad (15)$$

Model description In order to validate the methodology, the zone temperatures of three models are compared. The reference model is the IDEAS model with non-linear radiative heat transfer (Eq. 3 and 6), temperature-dependent interior convection (Eq. 11) and wind speed dependent exterior convection (Eq. 8).

The second model is identical to the reference model but it uses the linearised equations for the radiation and interior and exterior convection. The model is then fully linear except for its inputs.

Note that the linearisation of the exterior convection coefficient can cause a heat flow rate error of more to 150 W/m² due to the wide range of h_{cv} (from 7 to 55 W/m²K) and the potentially large difference between the ambient dry bulb temperature and the surface temperature. For the given example, the maximum deviation is 141 W/m². This error culminates when both wind speed and solar radiation are high, which causes both a high heat transfer rate and a high surface temperature.

The third model is the state space version of the second model. The SSM is loaded into Dymola using `Modelica.Blocks.Continuous.StateSpace`. Note that the difference between the third and the second model should be around the solver tolerance.

A comparison between the equation types and formats of the three models is given in Table 1.

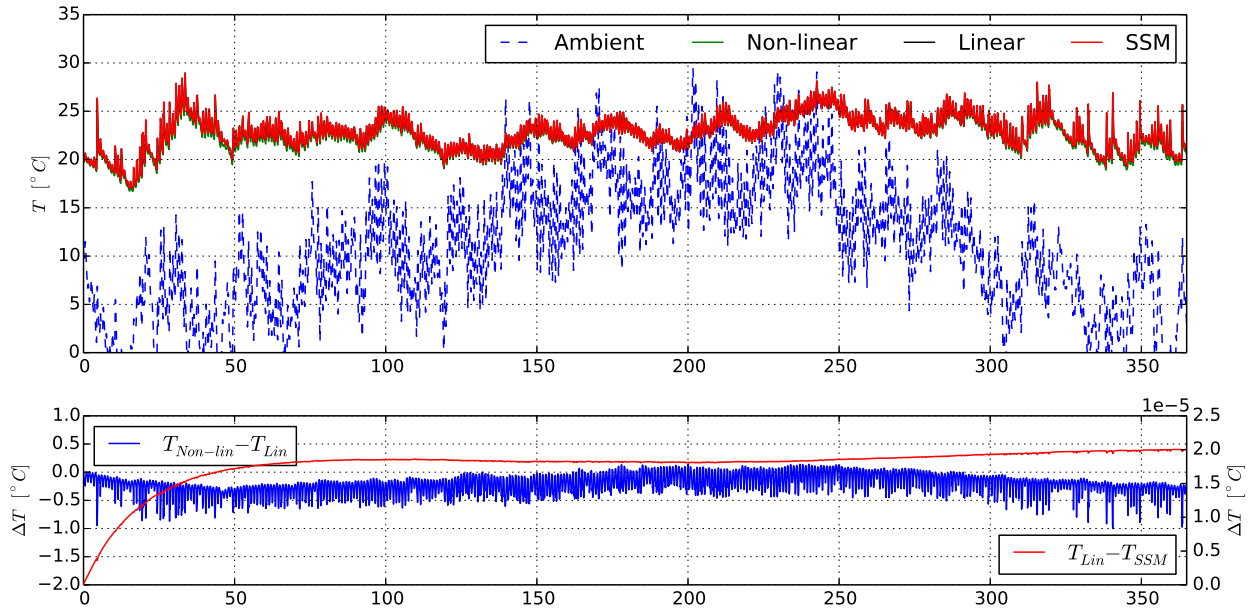


Figure 4. Result comparison between three model types for a one year simulation. The top graph shows absolute temperatures, while the other two graphs show absolute temperature differences.

Model comparison The three model versions are simulated in one model for a whole year using solver Dassl with a tolerance of 10^{-6} . The zone air temperatures are then compared. Figure 4 shows the southern zone temperature of the different models, the average error of the three zone air temperatures of the reference model and the linear model, and this average error for the linear model and its SSM. The figure shows that the zone temperature is excited over a realistic range. The CPU time is also compared¹. Normalized CPU times t_{norm} are computed by subtracting the ‘CPUtime’ required for a simulation that only computes the building model inputs. The CPU time ratio r_i is computed based on the non-linear reference case: $r_i = \frac{t_{norm,ref}}{t_{norm,i}}$. The total computation time for the reference case is 290 s.

Figure 4 shows that the linear model is a good approximation of the non-linear model as the absolute error remains smaller than 1K and its average is close to zero. This justifies the often made linear approximations in building modelling. Figure 4 also shows that the transformation of the linear model into a SSM does not introduce significant errors, as expected. This indicates that the model equations were successfully extracted by the `linearize2` function.

The linear model is faster than the non-linear model with $r_{lin} = 1.8$. This can be expected because linear equations typically require less operations and do not require non-linear algebraic loops to be solved. Interestingly, the SSM is much faster with a $r_{SSM} = 8.5$. This is

¹Simulations are performed using Dymola 2016 and Euler integration using a fixed time step of 10 s and a duration of 10^7 seconds. Euler integration is chosen to ensure that the same number of time steps is performed.

because the state space model contains only 50 states and therefore only 50 equations. The linear model contains 50 states and 453 additional² algebraic variables, which also need to be computed, often requiring the analytical solution of linear systems of equations.

These results suggest that the symbolic processing could be improved, resulting in faster models.

6 Model Order Reduction

The obtained SSM of Section 5 is accurate but a large number of states is used. This might be problematic for model-based optimal controllers such as MPC. In this section, we apply a MOR technique for different orders and we investigate their simulation accuracy compared to the original model. The comparison is extended by implementing a state observer for each ROM and by computing the 48-hours ahead prediction performance. The prediction performance is an indicator for the efficiency of the MPC which uses the model predictions to optimize the inputs of the system.

The different ROM’s are obtained by applying the Matlab function `reduce` to the SSM, using the default `balance` algorithm (`balancmr`). The simulation performance is compared using the minimum, maximum, mean and nominal root mean square error (NRMSE) (Eq. 16) between the original SSM and the ROM’s for each of the three zones. The errors are calculated over a period of 100 days. The applied heat inputs and the gains are computed as a sum of sinusoids with 30 frequencies and

²The translated linear model contains 453 ‘time-varying variables’ more than the translated SSM model.

realistic amplitudes. The weather-related inputs are computed using a typical year of Uccle (Belgium).

$$\text{NRMSE}^{(n)} = 100 \left(1 - \frac{\|y - \hat{y}^{(n)}\|}{\|y - \bar{y}\|} \right) \quad (16)$$

with y the output signal, \bar{y} its time averaged value, and $\hat{y}^{(n)}$ the output signal of the n order ROM.

Figure 5 shows the comparison for the reduced order models of different orders. For this particular example, the error rapidly decreases with the model order and it becomes negligible for ROM's with $n \geq 15$. The error on the south zone, which is irradiated by more direct sun light than the north zone, is the highest. We therefore conclude that the MOR technique can for this case successfully decrease the number of states without significant loss of accuracy but that a minimal number of states is necessary to correctly capture the faster dynamics of the system. These dynamics correspond to small thermal capacities of the different surfaces excited by the sun. This result was expected as the MOR technique typically removes the small eigenvalues of the system, responsible for the fast dynamics.

Note that by applying model reduction, the size of the SSM matrices decreases but the original matrices sparsity is lost. It is therefore not interesting to use reduced order model in Dymola as the number of additions and multiplication increase thereby. However, the loss of sparsity for optimal controller model is not an issue, since the required conversion from the continuous to the discrete time domain already removes that sparsity of the matrices.

7 Conclusion

This paper presents an approach for deriving linear state space models from BES models using the IDEAS library and Dymola. To this end, weakly non-linear equations are linearised. The remaining non-linear equations can be evaluated outside of the model since they do not depend on the model states. The resulting model is linearised using the Dymola function `linearize2`, which derives the state space matrices. The errors made by linearising the models are found to be acceptable. The SSM can be reduced using model order reduction techniques. For the tested case, the order of the model could be reduced by a factor three without significant loss of prediction accuracy. An important advantage of the presented methodology is that it automates the conversion of IDEAS BES models into state space formulation which can then be used for different purposes or by different programs.

The current implementation still presents some drawbacks that can be solved in the future. So far, the model can only have four different perpendicular orientations and all surfaces should either be horizontal or vertical.

Furthermore, the Medium in the zone should be simple air without any species concentration. Finally ventilation can only be modelled using heat flow inputs and not using mass/energy transport equations.

8 Acknowledgments

The authors acknowledge the financial support by the Agency for Innovation by Science and Technology in Flanders (IWT) for the PhD work of F. Jorissen (contract number 131012). The authors acknowledge the financial support by IWT and WTCB in the frame of the IWT-VIS Traject SMART GEOTHERM focusing on integration of thermal energy storage and thermal inertia in geothermal concepts for smart heating and cooling of (medium) large buildings. This work emerged from the Annex 60 project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme.

References

- S.A. Klein et al. TRNSYS 17, A Transient System Simulation Program. *Solar Energy Laboratory, University of Wisconsin, Madison, USA*, <http://sel.me.wisc.edu/trnsys>, 2010.
- H.B. Awbi and a. Hatton. Natural convection from heated room surfaces. *Energy and Buildings*, 30(3):233–244, August 1999.
- R. Baetens, R. De Coninck, F. Jorissen, D. Picard, L. Helsen, and D. Saelens. OpenIDEAS - An open framework for integrated district energy simulations. In *Submitted to Building simulation 2015*, Hyderabad, 2015.
- T. Defraeye, B. Blocken, and J. Carmeliet. Convective heat transfer coefficients for exterior building surfaces: Existing correlations and CFD modelling. *Energy Conversion and Management*, 52(1):512 – 522, 2011.
- E. U. Finlayson, D. K. Arasteh, C. Huizenga, M. D. Rubin, and M. S. Reilly. Window 4.0: Documentation of calculation procedures. Technical report, 1993.
- D. Gyalistras and M. Gwerder. Use of Weather and Occupancy Forecasts for optimal building climate control, two years progress report. Technical report, 2009.
- M. Kummert. *Contribution to the application of modern control techniques to solar buildings. Simulation-based approach and experimental validation*. PhD thesis, Fondation Universitaire Luxembourgeoise, 2001.
- B. Lehmann, D. Gyalistras, M. Gwerder, K. Wirth, and S. Carl. Intermediate complexity model for Model Predictive Control of Integrated Room Automation. *Energy and Buildings*, 58(0):250 – 262, 2013.

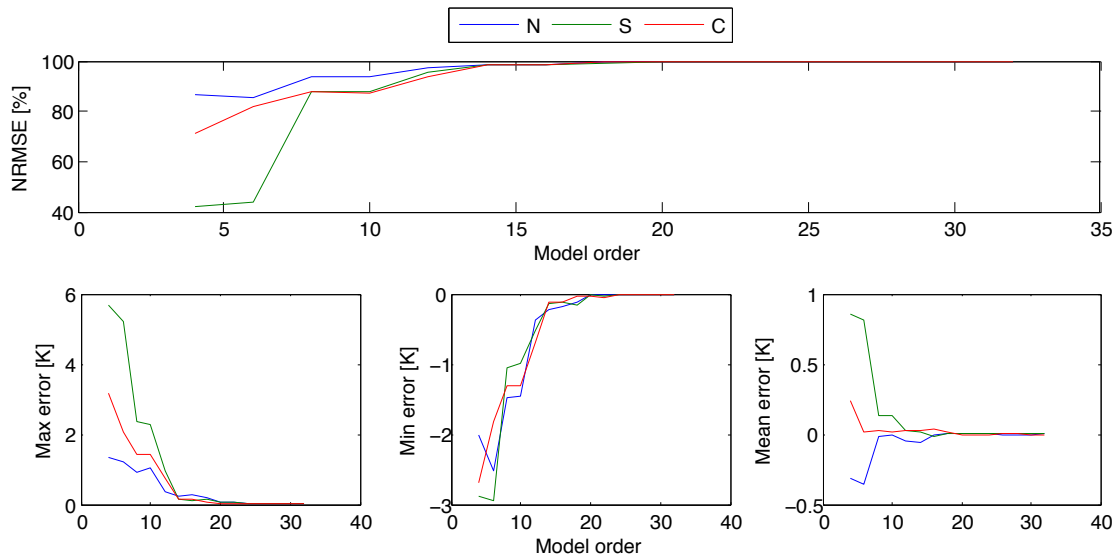


Figure 5. Nominal root mean square error (NRMSE), maximum, minimum and mean error between the original state space model of 50 states and the reduced order model of different order for the north (N), south (S) and corridor (C) zones.

R.J. Liesen and C.O. Pedersen. An evaluation of inside surface heat balance models for cooling load calculations. Technical report, American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc., Atlanta, GA (United States), 1997.

M. Otter. Modelica_linearsystems2, 2014. URL https://github.com/modelica/Modelica_LinearSystems2.git.

L. Perez-Lombard, J. Ortiz, and C. Pout. A review on buildings energy consumption information. *Energy and Buildings*, 40(3):394–398, 2008.

M. Sourbron, C. Verhelst, and L. Helsen. Building models for model predictive control of office buildings with concrete core activation. *Journal of Building Performance Simulation*, 6(3):175–198, 2013.

R. Strand, F. Winkelmann, F. Buhl, J. Huang, R. Liesen, C. Pedersen, D. Fisher, R. Taylor, D. Crawley, and L. Lawrie. Enhancing and Extending the Capabilities of the Building Heat Balance Simulation Technique for use in EnergyPlus. In *Proceedings of Building Simulation'99, Volume II*, pages 653–660, Kyoto, Japan, 1999.

D. Sturzenegger, D. Gyalistras, M. Morari, and Smith R. Semi-automated modular modeling of buildings for model predictive control. In *BuildSys 2012 - Workshop of ACM SenSys Conference, Toronto, Canada*, 2012.

D. Sturzenegger, D. Gyalistras, V. Semeraro, M. Morari, and R. Smith. BRCM Matlab Toolbox: Model Generation for Model Predictive Building Control. In *American Control Conference*, pages 1063–1069, Portland, June 2014.

C. Verhelst. *Model Predictive Control of Ground Coupled Heat Pump Systems for Office Buildings*. PhD thesis, KU Leuven, Belgium, 2012.

Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation

Filip Jorissen^{1,3} Michael Wetter² Lieve Helsen^{1,3}

¹Mechanical Engineering, KU Leuven, Leuven, Belgium, {filip.jorissen, lieve.helsen}@kuleuven.be

²Energy Technologies Area, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, mwetter@lbl.gov

³EnergyVille, Waterschei, Belgium

Abstract

This paper presents an approach for speeding up Modelica models. Insight is provided into how Modelica models are solved and what determines the tool's computational speed. Aspects such as algebraic loops, code efficiency and integrator choice are discussed. This is illustrated using simple building simulation examples and Dymola. The generality of the work is in some cases verified using OpenModelica. Using this approach, a medium sized office building including building envelope, heating ventilation and air conditioning (HVAC) systems and control strategy can be simulated at a speed five hundred times faster than real time.

Keywords: Modelica, speed, performance, buildings

1 Introduction

The Modelica language allows simulations of multidisciplinary problems. Combining multiple disciplines can lead to models that quickly grow in size and complexity. Consider for instance building energy modelling where building envelope, heating, ventilation and air conditioning (HVAC) systems and controls are integrated in a single model. The building envelope's thermal response typically has relatively slow dynamics, and heat transfer can be modelled using mostly linear equations. Building HVAC systems however contain a lot of non-linearities, performance curves and performance tables and typically have faster dynamics. Building control contains less dynamic components but contains a lot of discrete variables. Simulation of these types of systems can become very time consuming, limiting the use of these models.

Current literature does not provide a lot of insight into what determines computational speed and what Modelica users and library developers can do to speed up models. Chapter 14 of (Tiller, 2001) provides some hints on ways to improve computational performance such as using equations instead of algorithms, avoiding events, pro-

viding Jacobians of functions, selecting good solvers and tolerances and eliminating intermediate variables. The Dymola manual, section 5.7, suggests to limit overhead for writing results and to avoid chattering, and to use options such as inline integration and parallelization (Dassault Systèmes, 2014).

While the provided tips can be valuable, they are still high-level and often do not provide a lot of insight and consequently can be difficult to apply in practice. Also, a lot of potential for code optimization remains untouched. This paper provides insight in approaches to increase computational performance of models, specifically targeted at Modelica users and Modelica library developers.

Related research focuses on creating efficient solvers such as Quantized State System (QSS) solvers, using fast Jacobian evaluation techniques and using efficient parallelization strategies. These methods can be useful and complementary, but are outside of the scope of this work.

Firstly, some technical background about Modelica is given to allow easier interpretation of the discussion. Secondly, relatively small examples are used to demonstrate how Modelica code and models can be improved in Dymola and OpenModelica. These examples are based on the IEA-EBC Annex 60 Modelica library (Wetter et al., 2015) and are available online. Finally, the code improvements are applied to a large building model, demonstrating the potential of Modelica in conjunction with the solvers available in Dymola 2015 FD01 for whole building simulations.

2 Technical Background

The goal of this section is to provide the technical background required for understanding the analysis performed in this paper.

2.1 Governing Equations

A typical Modelica model can be mathematically expressed as an implicit system of Ordinary Differential

Equations (ODE) of the form

$$F(t, \dot{x}, x, u) = 0, \quad (1)$$

with initial conditions $x(0) = x_0$, where $F : [0, 1] \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$, for some $n, m \in \mathbb{N}$, t is time, x is the vector of state variables and u are inputs. For simplicity we omit discrete variables in this discussion. Often the equations can be manipulated analytically such that this system of equations can be expressed as an explicit ODE of the form

$$\dot{x} = \tilde{F}(t, x, u). \quad (2)$$

For example, if a heat capacitor with capacitance C is coupled to a fixed temperature boundary condition u through a thermal resistor R , then (2) becomes

$$\dot{x} = \frac{(u - x)}{RC}. \quad (3)$$

However, if the system of ODE is coupled to algebraic equations, as is common in building simulation, such a formulation is often not possible. In this case, the problem is defined by a system of Differential Algebraic Equations (DAE) of the form

$$\dot{x} = f(t, x, y, u), \quad (4)$$

$$0 = g(t, x, y, u), \quad (5)$$

with initial conditions $x(0) = x_0$, where $y \in \mathbb{R}^p$, for some $p \in \mathbb{N}$, are algebraic variables. Under certain smoothness assumptions and by use of the Implicit Function Theorem, one can show existence of a unique solution to (4) and (5) (Polak, 1997; Coddington and Levinson, 1955). This DAE can be solved by first solving (5) for y and then using y to compute \dot{x} . For example, consider a perfectly mixed volume with thermal capacity C and a pump that provides a constant pressure head $\Delta p = u_1$. Suppose that the pump provides water to the mixing volume with temperature u_2 and that the water mass flow rate $\dot{m} = y$ is defined by a simplified pressure drop equation describing a pipe as $\dot{m} = k \sqrt{\Delta p}$ or, equivalently, $y = k \sqrt{u_1}$. Equations (4) and (5) are then

$$\dot{x} = \frac{(u_2 - x) \cdot y \cdot c_p}{C}, \quad (6)$$

$$0 = y - k \sqrt{u_1}, \quad (7)$$

where c_p is the specific heat capacity of water and k is a constant.

2.2 Solution of Algebraic System

At time t , equation (5) needs to be solved for the algebraic variables y . Note that $g(\cdot, \cdot, \cdot, \cdot)$ consists of p equations $0 = g_i(\cdot, \cdot, \cdot, \cdot)$. Ideally, these can be reformulated using computer algebra and block-lower triangularization such that y can be explicitly computed.

However, such a reformulation is not always possible. In our example, the solution is still relatively easy since \dot{m} can be calculated directly from Δp , which is a known input. Δp may however be a function of an algebraic variable \dot{m} , for instance if a proportional controller is tracking a set-point for the mass flow rate. In this case an *algebraic loop* is created, with two equations needing to be solved simultaneously:

$$0 = \dot{m} - k \sqrt{\Delta p}, \quad (8)$$

$$0 = k_p \cdot (\dot{m} - \dot{m}_{set}) - \Delta p, \quad (9)$$

where k_p is the proportional gain of the P controller. Note that non-linear algebraic loops are typically more expensive to solve than linear systems of equations. Dymola will try to manipulate algebraic loops to limit the amount of work required for solving them. Information about the sizes of these (non-)linear systems before and after manipulation can be found in Dymola in the Translation tab under ‘Statistics’.

2.3 Time Integration

For simplicity, we explain the consequences of selecting explicit versus implicit time integration algorithms based on the Euler integration algorithm. Let the index i denote the current time step and consider a fixed step-size Euler integration method. The explicit Euler integration method computes

$$x_{i+1} = x_i + \Delta t \dot{x}_i = x_i + \Delta t f(t_i, x_i, y_i, u_i), \quad (10)$$

whereas the implicit Euler integration algorithm computes

$$x_{i+1} = x_i + \Delta t \dot{x}_{i+1} = x_i + \Delta t f(t_{i+1}, x_{i+1}, y_{i+1}, u_{i+1}). \quad (11)$$

Hence, for the implicit Euler algorithm, if $f(\cdot, \cdot, \cdot, \cdot)$ cannot be solved symbolically for x_{i+1} , an iterative solution is required to obtain x_{i+1} . This system of equations is large if there are many state variables. Solving it typically involves the calculation of the Jacobian and requires multiple iterations before convergence is achieved. This may lead to more work per time step, but it also allows large time steps being taken. Also, implicit integrators are better suited to solve stiff ODEs.

The Radau IIa integration is an implicit Runge-Kutta method. This method is a single-step method, meaning that the solution at the current time step is only affected by information from the previous time step. Integrators such as DASSL (Petzold, 1982) and Lsodar (Petzold, 1983; Hindmarsh, 1983) are multi-step methods (Dassault Systèmes, 2014). Multi-step methods use more than one previous value of the integrator’s solution to approximate the new solution. For a more detailed discussion on integrators we refer to Cellier and Kofman (2006) and Hairer and Wanner (2002).

2.4 Simulation Procedure

The simulation of a Modelica model typically proceeds as follows. First, the state variables are initialized based on the initial equations and start values. Then continuous time integration starts and results are saved at intermediate time intervals. At certain points in time, time or state events may occur, which need to be handled by the integrator. The equations $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$ that are solved can be found in the Dymola output file `dsmodel.mof` in the working directory. Output of this file can be enabled in the Translation tab. Note that no distinction between equations of $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$ is made in this file. The file may contain different sections that determine when the contained code is executed, such as the Initial section, Output section, Dynamics section, Accepted section and Conditionally accepted section. A description of these sections can be found in Dassault Systèmes (2014). Using `dsmodel.mof` and also the C-code in `dsmodel.c` can be important for debugging model stability and performance.

3 Analysis of Computational Overhead

This section builds upon the basic simulation procedure detailed above to provide further insight into reduction in computing time using illustrative examples. All numbered examples are available at <https://github.com/iea-annex60/modelica-annex60>, commit `e9e247d`, in the Modelica package `Fluid.Examples.Performance`. Presented results are based on Dymola 2015 FD01 and OpenModelica 1.9.3+dev (r25881) installed on Ubuntu 14.04 64 bit running on a virtual machine (Parallels 9.0.24251) on OS X Yosemite. Since the authors are most familiar with Dymola, all analyses are performed using Dymola, unless stated otherwise. A selection of results have been verified using OpenModelica to test their generality. Models that could not be compiled by OpenModelica were not verified.

The CPU time required for performing a simulation can be *approximated* by

$$t = \mathcal{O}(t_{init} + n_{fg} \cdot t_{fg} + n_{int} \cdot t_{int} + n_{data} \cdot t_{data}), \quad (12)$$

where t are the computation times of different steps, n are the number of times these steps are evaluated, and t_{init} is the time required to solve the initialization problem. The indices fg , int , $data$ refer, respectively, to the evaluation of functions $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$, the overhead for the integrator and the data storage.

The total computational overhead can be reduced by addressing any of these components. Knowing their values provides an important hint for where

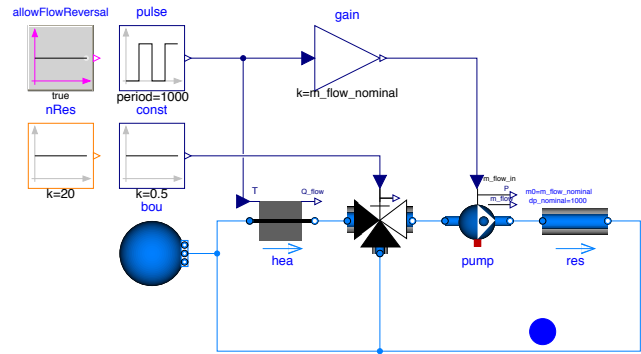


Figure 1. Example 1 illustration

computing time can be reduced. These values can be estimated from the Dymola simulation output. Setting `Advanced.GenerateBlockTimers = true` in Dymola generates the required output. The parameter n_{fg} in (12) equals the last column of the block timers. The value of t_{fg} equals the sum of column ‘Mean’ of rows ‘OutputSection’ and ‘DynamicsSection’. Row ‘Outside of model’ contains the overhead of the integrator, and possibly other overhead as well. n_{int} equals the ‘Number of (successful) steps’. n_{data} is determined by the settings in the ‘General’ and ‘Output’ tabs of the simulation settings.

Decreasing any of these factors will result in a lower simulation time. However it is not always clear how this should be achieved. A measure for decreasing one factor may also cause an increase in another. The following sections provide more insight into how to influence these different factors. Firstly the overhead for each function evaluation t_{fg} is discussed. Secondly the number of evaluations n_{fg} is discussed. Whenever possible, example models are provided based on the Annex 60 library. Finally a methodology is proposed for increasing the simulation speed of large building models.

3.1 Overhead per Evaluation

Evaluation of $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$ involves the evaluation of sequential code, algorithms, linear and non-linear algebraic loops, etc. We discuss how the overhead for this code can be reduced.

3.1.1 Algebraic Loops

When multiple equations are interdependent, an algebraic loop is formed. Depending on the type of equations the algebraic loop can be linear or non-linear. Solving non-linear algebraic loops requires iterative solutions such as encountered in a Newton-Raphson algorithm and is therefore more expensive. The user should therefore try to simplify or remove these systems where possible. We present some examples that demonstrate how this can be approached.

Algebraic Loops Iterating on Enthalpy Consider Example 1 shown in Figure 1. The presented hydraulic system contains a heater, a three-way valve and a pump setting the mass flow rate. The pump is connected to $nRes.k$ parallel pressure drop components res . The only two states are the temperatures of the heater and the pump with a time constant of 10 and 1 seconds, respectively. A pulsed signal sets the mass flow rate of the pump and the outlet temperature of the heater. The valve opening is set to 50%. The results are generated for $nRes.k = 20$ unless stated otherwise.

For the given configuration Dymola generates the following algebraic loops:

Sizes nonlinear systems of equations	{6, 21, 46}
Sizes after manipulation	{1, 19, 22}

Based on the C-code generated by OpenModelica, the following algebraic loops are generated:

Sizes nonlinear systems of equations	{7, 41, 47}
Sizes after manipulation	{1, 20, 23}

In Dymola, these algebraic loops can be analysed using the `dsmodel.mof` file. The first system solves for the mass flow rate in the left part of the fluid loop. The second system solves for the mass flow rate in the right part of the fluid loop. The third system solves for the enthalpies of the components in the right part of the fluid loop.

Dymola's BlockTimers generate the following output for the system dynamics:

Name of block,	Block,	CPU[s],
DynamicsSection:	14,	0.200, ...
Dynamics 2 eq:	15,	0.000, ...
Dynamics code:	16,	0.000, ...
Nonlin sys(1):	17,	0.007, ...
Dynamics code:	18,	0.000, ...
Dynamics 20 eq:	19,	0.066, ...
Dynamics code:	20,	0.002, ...
Nonlin sys(22):	21,	0.122, ...
Dynamics code:	22,	0.001, ...

Blocks 17, 19 and 21 clearly dominate the computational cost of this example. The Dymola file `dsmodel.c` shows that these block numbers correspond to the three non-linear systems. We explain how these systems can be simplified or removed.

The third system is created because there are no enthalpy states in the right circuit except in the pump. In general, the fluid can flow in both directions. Therefore the inlet and outlet enthalpies of all res components can be a function of all other res components, depending on the flow direction. This causes an algebraic loop since all enthalpy values depend on each other.

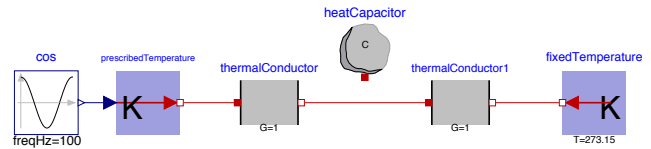


Figure 2. Example generating linear system of 2 equations

A common approach for decoupling algebraic loops is adding additional states (Zimmer, 2013). However, this can introduce fast dynamics, necessitating short time steps during parts of the simulation. The values of the state variables are solved by the integration algorithm, and hence they reduce the size of the algebraic loops. A simple example is shown in Figure 2 where a system of two linear equations is generated when the heat capacitor is unconnected. This system is decoupled when a heat capacitor is added, since the temperatures at the ports connecting the two conductances are then equal to the state variable of this heat capacitor and need no longer be obtained by solving an algebraic loop.

The enthalpy calculation of Example 1 can be simplified in a similar way by adding $nRes.k$ mixing volumes at the location of the blue dot in Figure 1, introducing a state in the flow path with a time constant for the enthalpy of 10 s. The state values for the enthalpy cause the system to become decoupled. The system size is now reduced from 46/47 to 4/7 before the manipulation, and from 22/23 to 1/3 after the manipulation for Dymola/OpenModelica, regardless of the value of $nRes.k$. Note that adding states also changes the simulation results.

In this example, a second approach is possible. We know that the fluid will always flow from the pump into the resistance. Therefore the inflow enthalpy of the resistances is always equal to the enthalpy leaving the pump. This knowledge can be passed on to the model by setting `allowFlowReversal=false` in the components where no flow reversal occurs. This causes the `min` and `max` attributes of the `m_flow` variable of the fluid ports to be set to zero. Dymola utilizes this and simplifies equations such as

```
H_out = semiLinear(port_a.m_flow,
                   inStream(port_a.h_outflow),
                   port_a.h_outflow)

into

H_out = port_a.m_flow * inStream(port_a.h_outflow)

or

H_out = port_a.m_flow * port_a.h_outflow .
```

It can conduct this simplification because the solver can now take into account that the mass flow rate will never become negative (or positive). Due to the simplified structure of the problem, the solver is able to sort the enthalpy equations in such a way that no algebraic loop is formed: the solver can evaluate the equations sequentially, following the fluid downstream starting from

	Successful steps	Jacobian evaluations	Function evaluations n_{fg}	Continuous time states	Mean time dynamics sec. [μ s]	Total time dynamics sec. [s]
N: Initial model	55	21	647	2	310	0.200
N: Enthalpy state	54	20	1448	22	103	0.150
N: No flow reversal	55	21	647	2	109	0.071
A: Enthalpy state	54	20	547	22	137	0.075
A: No flow reversal	55	20	557	2	116	0.065

Table 1. Solver output for 3 configurations of Example 1 (Figure 1), with $nRes.k = 20$ and analytic (A) or numeric (N) Jacobian

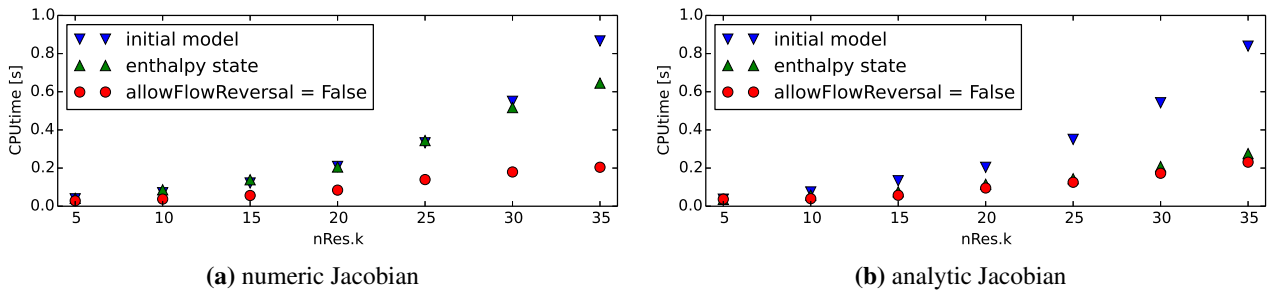


Figure 3. Simulation time for three variants of Example 1

known values of state variables. This causes the equations to be solved explicitly. OpenModelica does not make this simplification and consequently the algebraic loop size remains unchanged.

A different approach can be taken to break algebraic loops without relying on the solver to make simplifications. Many fluid components contain equations such as

```
port_a.h_outflow = inStream(port_b.h_outflow);
port_b.h_outflow = inStream(port_a.h_outflow);
```

which may be simplified into

```
port_a.h_outflow = if allowFlowReversal
                    then inStream(port_b.h_outflow)
                    else Medium.h_default;
port_b.h_outflow = inStream(port_a.h_outflow);
```

because the value of `port_a.h_outflow` should never be required for calculations upstream of `port_a`. Therefore it does not matter what its value is. Choosing a fixed value has the advantage that it allows breaking algebraic loops. Note that when the flow does reverse, the model equations will be wrong, which may cause unstable dynamics.

Figure 3a shows the influence of these two measures on the simulation time. Adding enthalpy states only reduced the computing time for $nRes.k > 20$. However, setting `allowFlowReversal=false` led to faster simulations. Note that the speed increase for the first case depends on the time constants of the new states. Larger time constants in general lead to faster simulations, but may introduce non-physical dynamics.

The first three rows of Table 1 allow analysing the results in further detail. Both measures allow reducing the computational work for each evaluation of f and g in the `dynamics` section from 310μ s to $\sim 106 \mu$ s. The overall speed when using `allowFlowReversal=false` is however better due to the lower number of

function evaluations that is required: 647 instead of 1448. The increased number of function evaluations is caused by the increased number of states in the model. It turns out that the higher number of state variables leads to significantly more function evaluations, probably because by default, Dymola computes a *numerical approximation* to the Jacobian based on numeric differentiation.

Due to the performance penalty for approximating the Jacobian, the simulations are repeated using an analytic Jacobian, which can be done in Dymola by setting `Advanced.GenerateAnalyticJacobian=true`. In OpenModelica, an option for this exists in the simulation setup. Results are shown in Figure 3b and in Table 1. The penalty for adding new states is almost completely removed when using an analytic Jacobian. Somehow the average execution time for the dynamics section increased slightly, even though the equations did not change. The reason for this is unclear. The results indicate that the analytic Jacobians should be used whenever possible, especially for models with a large amount of states.

From this analysis we conclude that the user should be cautious when adding states for decoupling algebraic loops. If they are added, setting `Advanced.GenerateAnalyticJacobian=true` may reduce computing time. An alternative approach is to use physical insight to simplify the equations where possible, in a way similar to setting `allowFlowReversal=false`. Also, it may be beneficial to remove the states that are added by default in three-way valves and other components containing mixing volumes. This can be done by setting `energyDynamics=massDynamics=SteadyState`. Most likely this change will create larger systems, but often these can be simplified using the approach explained above.

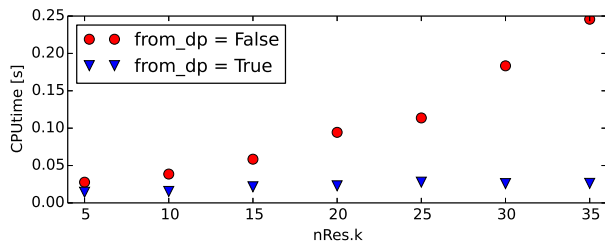


Figure 4. Example 1 illustrating computation time for solving mass flow rates through parallel resistances

Algebraic Loops Iterating on Mass Flow Rates and Pressures When setting `allowFlowReversal=false`, the remaining computation time is almost entirely used for computing the mass flow rates and pressures. We now focus on reducing this computing time further.

The pressure drop equations in this non-linear system can be written either as $\dot{m} = f(\Delta p)$ or as $\Delta p = f^{-1}(\dot{m})$ for some function $f(\cdot)$ or its inverse $f^{-1}(\cdot)$. The value of the parameter `res.from_dp` will pick one or the other formulation. If `from_dp=false`, then the system has size 21/22 before and 19/20 after manipulation, otherwise it has sizes 21/22 and 1/1 in Dymola/OpenModelica. This can be explained as follows. When `from_dp=true`, the mass flow rate is calculated as a function of the pressure difference Δp . Therefore Δp is chosen as an iteration variable. The symbolic processing algorithm detects that all resistances are in parallel and hence must have the same pressure drop. Therefore, they can all use the same iteration variable, leading to a much smaller system. This leads to a significant speed improvement, as shown in Figure 4.

Example 1 uses a pump which sets the *mass flow rate* to an input value and which is connected to `nRes.k` *parallel* pressure drop components. The solver can exploit the system structure by selecting the common pressure drop as an iteration variable. The “dual” problem (Example 2) could be to consider a pump which takes the *pressure drop* as an input value and which is connected to `nRes.k` pressure drop components connected in *series*. In this case, it is advantageous to set `from_dp=false` since Dymola and OpenModelica then select the common *mass flow rate* as the iteration variable, as illustrated in Figure 5.

These were fairly simple problems. In practice, combinations of parallel and series connections are used, making the choice of the parameter `from_dp` difficult. However, it is often possible to aggregate multiple pressure drop components that are connected in series. If all components have the same nominal mass flow rate `m_flow_nominal`, then the nominal pressure drops `dp_nominal` can be added into one component, reducing the series branch into a single pressure drop equation. Otherwise `dp_nominal` needs to be rescaled. This ap-

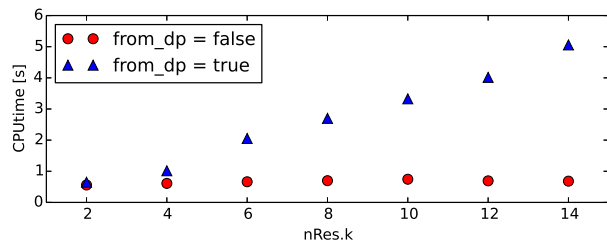


Figure 5. Example 2 illustrating computation time for solving mass flow rates through resistances in series

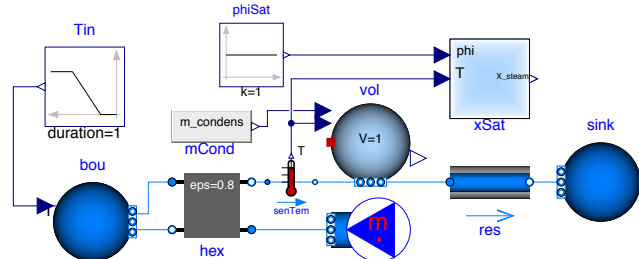


Figure 7. Example 4 illustration

proach can also be used when a valve is connected in series to the pressure drop components. The valve parameter `dpFixed_nominal` should then be used.

Figure 6a shows Example 3 where `nRes.k` parallel instances of a series connection of two resistances are simulated. The simulation time for this example is shown in Figure 6b. The parameter `mergedDp` indicates whether the two resistances are merged into one. Merging the two resistances gives much better results, especially when combined with `from_dp=true`. However when the two resistances are not merged, it is better to set `from_dp=false`.

Model Design for Avoiding Algebraic Loops Developers should avoid coupling systems of equations that are only weakly dependent. Consider for instance the model of a condensing heat exchanger. Such a model contains equations for the pressure drop, heat flow rate and water vapour condensation. One should try to avoid coupling these equations into one algebraic loop.

Example 4 in Figure 7 shows a simple condensing heat exchanger model. Along the flow path, first air cools in the heat exchanger `hex`, then condensate is extracted from the stream in `vol` (steady state) and finally the remaining mass is sent through a pressure drop component. Ideally the solver would be able to first compute the mass flow rate based on the pressure drop characteristic. Using this mass flow rate, the heat flow rate can be computed since it only depends on inlet temperatures and mass flow rates. Finally moisture can be extracted such that the air stream becomes saturated. In practice this sequential calculation is not possible because removing water vapour from the air affects its mass flow rate and therefore also the pressure drop. As a consequence the equations for the

Obsolete Model Variables In some cases it may be wise to eliminate model variables. Consider for instance variables a , b and c where $b = 2a$ and $c = 2b$. If b is not used in any other equation, then it is better to write $c = 4a$ and remove b .

It may be important to analyse the effects of such changes in detail. Consider for instance the model of a discretised wall. The model consists of a series of temperature states with an adiabatic boundary condition on one side and a sinusoidal temperature on the other side. Typically, this will be modelled using thermal capacitances C and thermal resistors R . A Modelica implementation could be as presented by Example 6.

```
model Example6
  parameter Integer nTem = 500;
  parameter Real R = 0.001;
  parameter Real C = 1000;
  Real[nTem] T;
  Real[nTem+1] Q_flow;
equation
  Q_flow[1] = ((273.15+sin(time))-T[1])/R;
  der(T[1]) = (Q_flow[1]-Q_flow[2])/C;
  for i in 2:nTem loop
    Q_flow[i] = (T[i-1] - T[i])/R;
    der(T[i]) = (Q_flow[i]-Q_flow[i+1])/C;
  end for;
  Q_flow[nTem+1] = 0;
end Example6;
```

In this model variables Q_flow are calculated but not necessarily needed. These variables can be eliminated as illustrated in Example 7.

```
model Example7
  parameter Integer nTem = 500;
  parameter Real R = 0.001;
  parameter Real C = 1000;
  parameter Real tauInv = 1/(R*C);
  Real[nTem] T;
equation
  der(T[1]) = ((273.15+sin(time))-2*T[1]+T[2])
    *tauInv;
  for i in 2:nTem-1 loop
    der(T[i]) = (T[i-1]-2*T[i]+T[i+1])*tauInv;
  end for;
  der(T[nTem]) = (T[nTem-1]-T[nTem])*tauInv;
end Example7;
```

Comparing Example 7 to Example 6 a variable has been eliminated but the number of operations within the `for` loop remains the same. In particular, there are two additions and two *divisions* in Example 6, and two additions and two *multiplications* in Example 7. However, Example 7 is $\sim 83\%$ faster in Dymola (2.83 s \rightarrow 0.49 s) and OpenModelica (9.2 s \rightarrow 1.6 s). It turns out that this is mostly because a division generates more overhead than a multiplication, probably because of guarding against division by zero. This performance penalty can be reduced significantly by adding annotation(`Evaluate=true`) to parameters R and C , or by creating a dummy parameter similar to `tauInv` and by multiplying with this parameter. This reduces simulation time to 0.65 s $>$ 0.49 s in Dymola and 2.39 s $>$ 1.6 s in OpenModelica.¹ The reason for the remaining

performance difference is unclear but may be explained by the extra variables Q_flow , which may generate overhead.

From this analysis we conclude that there exists unexploited code optimization potential in popular Modelica tools. Certain variables can be eliminated and dummy parameters can be introduced to avoid parameter divisions during each time step. Until these issues are resolved, users can avoid performance penalties by taking into account these limitations by reformulating models.

Duplicate Code The developer should avoid making models that generate duplicate code. A good example is a window model, which requires the solar irradiance to be calculated. Since this calculation is influenced by parameters such as the window orientation and inclination angle, the developer may choose to include these equations in the window model. If multiple windows have the same orientation and inclination, then this means that the same calculation is repeated multiple times. This is not necessarily a problem if the overhead is small. However, in the case of a window model, the computation involves a lot of trigonometrical calculations and it would be better to isolate this calculation in a separate model. An example implementation of this problem can be found in the IDEAS library (Baetens et al., 2015). However, putting the solar irradiation in a separate model requires the user to keep the radiation computation consistent among multiple models.

An illustration of common subexpression elimination is given by Example 8.

```
model Example8
  Real a = sin(time+1);
  Real b = sin(time+1);
end Example8;
```

The Dymola C-code evaluates the sine and addition only once:

```
W_[0] = sin(Time+1);
W_[1] = W_[0];
```

This simplification is not made in OpenModelica since it evaluates the $\sin(\cdot)$ function once for a and once for b .

Still, more complicated common subexpressions such as in IDEAS are not detected by both tools. Therefore, improving the common subexpression elimination would allow further performance improvements.

3.2 Number of Evaluations

The previous section focussed on how to reduce the computational overhead for each evaluation of $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$. The current section focusses on how to reduce the *number* of evaluations. Important aspects are the time constants of the system, the system stability, the number of events, computing the Jacobian and the integrator choice.

¹These CPU times are based on the total Dynamics section time in Dymola and the ‘simulation’ timer in the Statistics output of Open-

Modelica when performing 100 000 Euler integration steps of Example 6 and Example 7.

System Time Constants When a system has fast dynamics, then the solver has to track these dynamics with small step sizes. In general, systems with large time constants have shorter calculation times. It may therefore be advantageous to make certain dynamics slower, especially the fastest dynamics in the system. Dymola option “Which states that dominate error” may be used to identify these states. Changing the dynamics may however be non-physical or introduce instability in feedback control loops. In this case a different option may be to remove the fast dynamics completely and simulate the system as a steady state system. Note, however, that this may increase the size of the algebraic system of equations.

The latter approach may be very effective when considering air flow networks. If air is modelled as compressible, pressure states are created in instances of `MixingVolume`, unless `massDynamics=SteadyState`. These states however introduce small time constants if part of a building air flow network. It may therefore be better to remove them. Again, this may create larger systems of equations.

System Stability If a feedback control loop is tuned badly, oscillatory behaviour can occur. A variable time step integrator may track these oscillations, leading to a major decrease in simulation speed. Note that it may be difficult to see these oscillations when the output interval is set too large.

Number of Events Events require the integration to stop and restart, typically with a lower order method and with smaller time steps. In addition, for state events, typical ODE solvers require an iterative solution to find the time when the event happens.

Computing the Jacobian Some integrators require the Jacobian to be calculated. Having more states leads to a larger Jacobian, as was illustrated in Example 1. Since by default, Dymola and OpenModelica use numeric differentiation to approximate the Jacobian, a lot of finite differences need to be calculated, each requiring a function evaluation. Note that in particular models with a larger number of states benefit more from having an analytic Jacobian, since the number of Jacobian entries equals the square of the number of states.

Integrator Choice Many integrators use an implicit integration scheme. This typically requires the computation of a Jacobian and requires iterations to be performed before reaching convergence. This can lead to more function evaluations. However, for stiff systems, implicit integrators are more efficient than explicit integrators.

3.3 Analysis of Large Problems

In the previous sections, computing time was analysed using small models. In building simulation, models can however become considerably larger and analysing the computational speed can be difficult since it depends on a lot of factors, including the unknown solver implementation. Still, we predict some trends for the computation time, based on the size of the model.

Consider a model of a district energy system, including building models and an electrical grid. When doubling the size of the district, ideally the computational time would double as well, such that computational time scales linearly with problem size. Let us analyse this further based on Equation 12. Ideally t_{fg} scales linearly with the problem size. In practice this is not necessarily the case. The electrical grid of the district typically results in a large non-linear system of equations since all electrical components have very fast transients and are therefore modelled as steady state components. Doubling the size of the model therefore also doubles the size of the algebraic loop. Example 1 has shown that computational time for algebraic loops does not scale linearly with size and therefore larger models will become computationally slow. Equations outside algebraic loops can be solved sequentially. Therefore their computational time does scale linearly.

Because t_{fg} scales, at best, linearly with size, n_{fg} should remain constant if we want to obtain overall linear scaling of the computational time. However, firstly, generally n_{fg} also grows with problem size, for example because larger problems have more controllers that may trigger events. If the amount of buildings doubles, then the amount of state events may double, which causes a performance penalty. Secondly, when a numeric Jacobian needs to be computed, then n_{fg} will increase since the number of states increases linearly with the problem size. The number of operations for an implicit integrator typically does not scale linearly either. Solving dense implicit systems typically requires $\mathcal{O}(n^3)$ operations (Hairer and Wanner, 2002). Building model Jacobians are however very sparse. It is not clear how well this is exploited by Dymola. An integrator such as `Rkf1x4` can have an operation count that is linear with the problem size, unless the fixed time step is changed. For certain large problems that do not require event handling, it can therefore be advantageous to use these simple integrators, also because they do not require a Jacobian to be calculated.

3.3.1 Parallelization

Dymola supports parallelization for the calculation of $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$ (Dassault Systèmes, 2014) and analytic Jacobian (see `Advanced.ParallelizeAnalyticJacobian`).

However parallelization generates overhead for syn-

Integrator	Tolerance / step size	CPUtime [s]	Dynamics section [s]	Outside of model [s]	Function evaluations n_{fg}	State events	Time events	Jacobian evaluations	E_{el} [error]
Dassl	1 E-6	4261	3538	476	787341	41	8	1235	-4.35 E-6
Dassl	1 E-4	3088	2759	327	546326	36	8	862	3.17 E-3
Radau IIa	1 E-6	4042	2400	1416	453073	37	8	347	1.64 E-3
Lsodar	1 E-6	3450	2666	547	679486	44	8	1047	-4.35 E-6
Lsodar	1 E-4	2073	1435	515	347018	41	8	537	-2.25 E-5
Lsodar	1 E-2	1655	1152	406	256458	38	8	399	4.51 E-3
Dopri45	1 E-6	194	159	17.0	41166	39	8	0	4.68 E-4
Dopri45	1 E-8	199	162	18.3	42017	39	8	0	1.96 E-6
Rkfix4	20 s	15.4	11.3	1.1	2717	39	8	NA	1.34 E-2
Rkfix4	5 s	50.6	42.9	1.5	10233	43	8	NA	2.52 E-3
Rkfix4	1 s	224	202	3.2	50211	38	8	NA	1.28 E-3
Euler	5 s	24.0	18.2	1.7	4271	50	8	NA	-2.00 E-3
Euler	0.25 s	446	389	12.8	80233	41	8	NA	-4.22 E-4

Table 2. Example building model statistics for various integrators and tolerance options. Results are the solution statistics (when available, else ‘NA’) and the relative error of E_{el}

chronization and communication. The authors have not been able to gain notable improvements in simulation speed in building applications by using parallelization in Dymola 2015 FD01.

3.3.2 Example of Large Building Model

The approach explained in this paper was applied to a building model based on a real case (Solarwind, Luxembourg), containing 32 IDEAS (Baetens et al., 2015) building zones with individual concrete core activation circuits (Baetens et al., 2015) and Variable Air Volume (VAV) boxes including heating battery, bore field model (Picard and Helsen, 2014), solar collector (Wetter et al., 2014), four thermal storage devices (Wetter et al., 2014), one pellet boiler, four heat pumps (Baetens et al., 2015), two adiabatic/active heat recuperating air handling units, pumps (Wetter, 2013) and valves (Wetter et al., 2015) and a control strategy based mostly on hysteresis controllers, PID controllers, heating/cooling curves and boolean algebra. The model has 2468 continuous time states and 28342 time-varying variables.

Special care was taken to make sure that the smallest time constants are in the order of 30 s. Therefore air ducts are steady state, pumps and valves have no opening delay or filter and pipes were lumped into only a few states per circuit branch, thereby allowing to increase the time constant. Temperature sensors are assumed to have a time constant in the order of one minute. Using dynamic sensors avoids coupling the thermal equations with the control equations into a single algebraic loop.

This model was simulated for $t_{end} - t_{start} = 10\,000$ s using various implicit integrators, with numeric Jacobians and explicit integrator `Dopri45`. The total amount of function evaluations exceeds 40 000 in each case. This is on average one function evaluation every 0.25 s, while the smallest time constant of the system is ~ 30 s. Therefore it makes sense to use an explicit fixed step integrator. Table 2 shows the results, including fixed step ex-

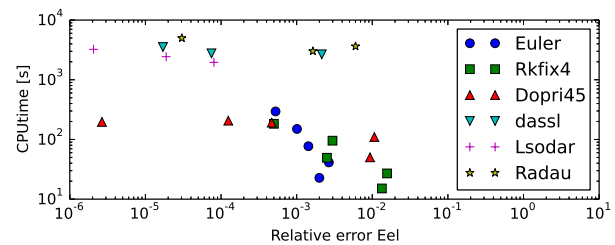


Figure 8. Relative errors of E_{el} for various solvers and tolerances or fixed time step sizes

PLICIT integrators `Rkfix4` and `Euler`. It contains statistics and the error on one simulation result that is of interest, namely the integrated electrical power consumption of the building E_{el} . The relative error was calculated using `Dassl` with a tolerance of 10^{-8} , which produced a result of 4.591880 kWh.

From these results and Figure 8 it is clear that implicit integrators are very slow compared to explicit integrators for this problem. Fixed step methods are especially fast when high accuracy is not required, allowing a simulation speed 500 times faster than real time, which is more than 100 times faster than `Dassl`. For higher accuracies, `Dopri45` can be used. `Lsodar` is the fastest implicit integrator that was tested. Note that the simulation can easily be made faster by using a larger step size, at the cost of accuracy. Also, using larger step sizes will eventually lead to numerical instabilities. The user may therefore want to adjust the dynamics of the system, or set certain dynamics to steady state. It is thus important that models expose these parameters and allow easy configuration.

The achieved speed increase is considerable. However, this still requires about 18 hours for a one-year simulation. As this is longer than typical building energy simulations, we think that further research is desirable to reduce the simulation time further.

4 Conclusion

We conclude that the analysis of algebraic loops, the optimization of Modelica code and the application of physical insight can lead to significant simulation time improvements. Analysis of the model time constants, avoiding system instabilities, using analytic Jacobians and proper integrator choice can also be important. These modifications were applied to a large building model where removal of all ‘fast’ dynamics allowed explicit integrators to perform well. Fixed step integrators can also be used if simulation results do not need to be very accurate. `Euler` integration performs very well in terms of computation time, allowing detailed office building simulations at a speed 500 times faster than real time.

Further work can focus on analysing and changing the problem structure in such a way that parallelization can be used efficiently. It should also be investigated up to which extent models can be made faster by changing the model dynamics, which allows larger time steps to be taken, without introducing too large errors. The proposed changes demonstrate that further symbolic processing in Dymola and OpenModelica is possible. We also propose to use analytic Jacobians by default for all Jacobian *elements* where an analytic Jacobian can be computed.

5 Acknowledgements

The authors acknowledge the financial support by the Agency for Innovation by Science and Technology in Flanders (IWT) for the PhD work of F. Jorissen (contract number 131012).

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.

This work emerged from the Annex 60 project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme. Annex 60 will develop and demonstrate new generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and BIM standards.

References

Ruben Baetens, Roel De Coninck, Filip Jorissen, Damien Picard, Lieve Helsen, and Dirk Saelens. Openideas - an open framework for integrated district energy simulations. In *Building simulation 2015, submitted*, Hyderabad, 2015.

François E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer US, 2006.

Earl A. Coddington and Norman Levinson. *Theory of ordinary differential equations*. McGraw-Hill Book Company, Inc., New York-Toronto-London, 1955.

Dassault Systèmes. Dymola user manual, vol. 1, 2014.

Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer-Verlag Berlin Heidelberg, 2002.

Alan C. Hindmarsh. Odepack, a systematized collection of ode solvers. *IMACS transactions on scientific computation*, 1:55–64, 1983.

Linda R. Petzold. Description of dassl: a differential/algebraic system solver. Technical report, Sandia National Labs., Livermore, CA (USA), 1982.

Linda R. Petzold. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM journal on scientific and statistical computing*, 4(1): 136–148, 1983.

Damien Picard and Lieve Helsen. Advanced Hybrid Model for Borefield Heat Exchanger Performance Evaluation, an Implementation in Modelica. In *10th International Modelica Conference 2014*, pages 857–866, Lund, 2014.

Elijah Polak. *Optimization, Algorithms and Consistent Approximations*, volume 124 of *Applied Mathematical Sciences*. Springer Verlag, 1997.

Michael Tiller. *Introduction to Physical Modeling with Modelica*. Springer US, 2001.

Michael Wetter. Fan and pump model that has a unique solution for any pressure boundary condition and control signal. In Jean Jacques Roux and Monika Woloszyn, editors, *Proc. of the 13-th IBPSA Conference*, pages 3505–3512, 2013. URL <http://simulationresearch.lbl.gov/wetter/download/2013-IBPSA-Wetter.pdf>.

Michael Wetter, Wangda Zuo, Thierry S. Noudui, and Xiufeng Pang. Modelica buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014.

Michael Wetter, Marcus Fuchs, Pavel Grozman, Lieve Helsen, Filip Jorissen, Moritz Lauster, Dirk Müller, Christoph Nysch-Geusen, Damien Picard, Per Sahlin, and Matthis Thorade. IEA EBC annex 60 modelica library - an international collaboration to develop a free open-source model library for buildings and community energy systems. In *Building simulation 2015, submitted*, Hyderabad, 2015.

Dirk Zimmer. Using Artificial States in Modeling Dynamic Systems : Turning Malpractice into Good Practice. In *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 77–85, 2013.

Energy Efficient Design for Hotels in the Tropical Climate using Modelica

Reymundo J. Miranda^{1,2} Sen Huang¹ German A. Barrios¹ Dan Li¹ Wangda Zuo^{1*}

1. Department of Civil, Architectural and Environmental Engineering, University of Miami, Coral Gables, Florida, USA,

2. UCI Engineering, Miami, Florida, USA

rmiranda@ucieng.com, {s.huang10, g.barrios, d.lil1}@umiami.edu, w.zuo@miami.edu

Abstract

For hotels located in the tropical climate, a significant amount of energy is attributed to the domestic hot water (DHW) usage and the space cooling. To improve the energy efficiency of hotels in the tropical climate, we proposed a heat recovery system that could utilize the waste heat from the space cooling system to pre-heat the city water supplied to the DHW system. To support the system design, we selected Modelica to model the heat recovery system and its control, which is difficult to be simulated by conventional building simulation tools. The Modelica *Buildings* library and the *Modelica_StateGraph2* library were employed to build the system model. A hotel in Miami, Florida, U.S. was selected for the case study. The simulation results showed that the proposed heat recovery system could save up to around 30% boiler energy use in the DHW system.

Keywords: energy efficient design, hotel, tropical climate, Modelica

1 Introduction

In the U.S., hospitality facilities, such as hotels and resorts, account for 7% of the primary energy consumption of all commercial buildings (U.S. Department of Energy), which is equivalent to approximately 1.3% of primary energy consumption in the nation. In an average hotel, the Heating, Ventilation and Air Conditioning (HVAC) system accounts for around 50% of electricity usage and up to 86% of natural gas consumption (U.S. Environmental Protection Agency, 2008). Due to their significant energy consumption, improving energy efficiency of the HVAC systems in hospitality facilities is of great interests to the society.

The conventional HVAC system for the hospitality facilities consists of two parts: the Domestic Hot Water (DHW) system and the space conditioning system. The DHW system provides the hot water to the kitchen and the guestroom. It obtains the supplement water from the municipal water network, which is called “city water”. The city water is then heated by heating equipment such as boilers. The space conditioning

system provides cooling \ heating to the space. In the cooling condition, the heat from the building is extracted by the space conditioning system and usually dumped to the ambient environment. The dumped heat is called “waste heat”. In the heating condition, the space condition system extracts heat from the ambient environment or boilers and then injects the heat into the hotel space.

The same HVAC systems are also implemented for hotels in the tropical climate. However, the space cooling in tropical climate is the dominant usage of the space conditioning and there is a significant amount waste heat generated from the space cooling all the year. Thus, it is possible to save the energy consumption by the heating equipment in the DHW system if we can recover the waste heat from the space conditioning system and use it to preheat the city water before it enters into the heating equipment.

In addition, if there is a capacity control in the heat rejection of the space conditioning system (e.g. variable speed fan control in the cooling tower), we can also reduce the energy consumption by the heat rejection since the waste heat injected by the heat rejection system decreases if the heat recovery occurs. Besides the energy saving, we also have other benefits from the heat recovery: it can extend the life of both the heating equipment and the heat rejection system by reducing their usage. If the cooling towers are employed in the heat rejection system, the heat recovery can also further help the environment by reducing the usage of evaporative cooling at the cooling towers, which requires make-up water and chemicals for treating the make-up water.

On the other side, there are some potential negative impacts associated with the heat recovery: adding an extra piping system to connect the space conditioning system and the DHW systems requires additional initial cost in equipment, such as pipes, heat exchangers, pumps and valves, and labor for installation. It will also need additional pump energy to cycle water between two systems to enable the heat recovery. Finally, controlling the heat recovery can be a challenge since the system operates under various conditions with complicate flow loops.

Overall, the heat recovery seems to be promising since it is not just economically beneficial but environmentally as well. However, to find a balance in terms of costs and benefits, it is necessary to quantitatively evaluate the performance of the proposed heat recovery design.

In this paper, we presented our research in designing and modeling a heat recovery system for the Grand Beach Hotel in Miami, Florida, U.S. We first introduce the detailed design of the heat recovery system. Then we evaluate the energy performance of the heat recovery system with simulation. In the simulation, Modelica was used to establish the system model. After showing the simulation results of different operating scenarios, we discuss the future work for the project.

2 Design of the Heat Recovery System

As shown in Figure 1 and Figure 2 are the DHW system and the space condition system of the Grand Beach hotel, respectively. The DHW system contains a group of three identical boilers with a combined total capacity of 350kW and they are represented by a single boiler (Boiler-1), a group of three identical domestic hot water tanks with a total capacity of 3000 L and they are also represented by a single DHW tank (HW Tank). The DHW system provides hot water at 60°C. Part of the 60°C hot water would be directly supplied to the kitchen and the rest would be mixed with the city water to provide a 43.3°C hot water to the guestroom. To ensure the quick delivery of the hot water, the hot water is continuously circulating within the distribution network. The space conditioning system is made up of two heat pumps, two cooling towers, one heat exchanger and one small (102.4kW) boiler (Boiler-2). Boiler-1 is the dedicated main boilers that provide the heat for both the DHW system and the space conditioning system. Boiler-2 is a backup boiler that operates only when Boiler-1 is not able to meet the heating demand.

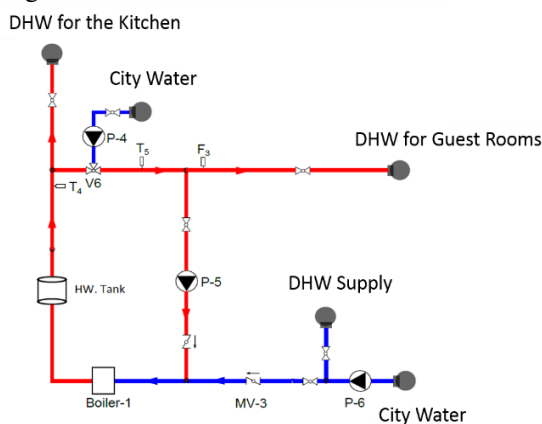


Figure 1. Schematic of the DHW system

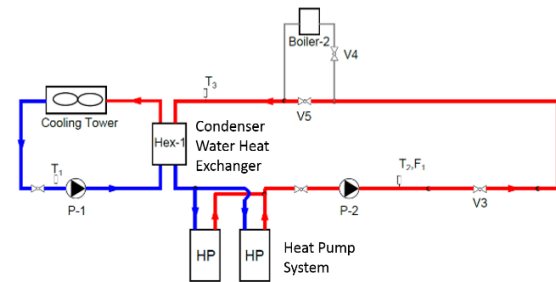


Figure 2. Schematic of the space conditioning system in the space cooling mode

As mentioned in the introduction, we can save the energy consumption of both the DHW system and the system conditioning system by recovering the waste heat from the latter to the former. In fact, Miami provides an ideal environment for the use of the heat recovery. The climate in Miami is considered to be tropical or sub-tropical. According to the ASHRAE standard 169 (ASHRAE, 2006), Miami is located in Climate Zone 1A, which is very hot and humid. Thus, there is a large amount of waste heat from the space cooling throughout the year. With that in mind, we proposed to use a connection loop, which recovers the heat from the space conditioning system to the DHW system through a heat exchanger (Hex-2 in Figure 3a). There are seven possible operating states for the whole system varying from the space heating to the space cooling. In the winter of Miami, there are occasionally a few cold days, especially during the morning that the space heating is needed. Since hotel guests tend to take shower in the morning, the DHW demand may also be high at the same time when the space heating is needed. If the DHW demand is extremely large that requires the full capacity of Boiler-1, the space conditioning and the DHW systems will run independently (State 1). The purpose is to guarantee the supply of the DHW. In this case, the heat recovery system will stop working.

When the DHW demand drops and Boiler-1 has additional capacity to meet a partial demand of the space heating, the system will operate at State 2 that use both Boiler-1 and Boiler-2 for the space heating through the connecting loop. This is anticipated to be the typical operating state for the space heating. The flow direction in the connection loop at State 2 is shown in Figure 3a.

If the combined demand for the space heating and the DHW drops to a level that can be met by Boiler-1, Boiler-2 will be turned off and the system will operate at State 3.

State 4 happens when there is no need for the space heating or cooling, which seldom happens. At this state, the DHW system and the system conditioning system are disconnected and no space heating or cooling will be provided. This state is used as a transitional period between the space heating and the space cooling.

At State 5, a moderate space cooling and a large amount of the DHW is provided at the same time. The waste heat from the space cooling is used to heat the cool city water before it is further heated by Boiler-1. The demand for the DHW is sufficiently large so that the DHW subsystem can absorb all the waste heat from

the space conditioning system. If the DHW subsystem cannot absorb all the waste heat from the space conditioning system because either the large demand for the space cooling or insufficient demand for the DHW, the cooling towers are kicked on to eject the remaining waste heat to the ambient

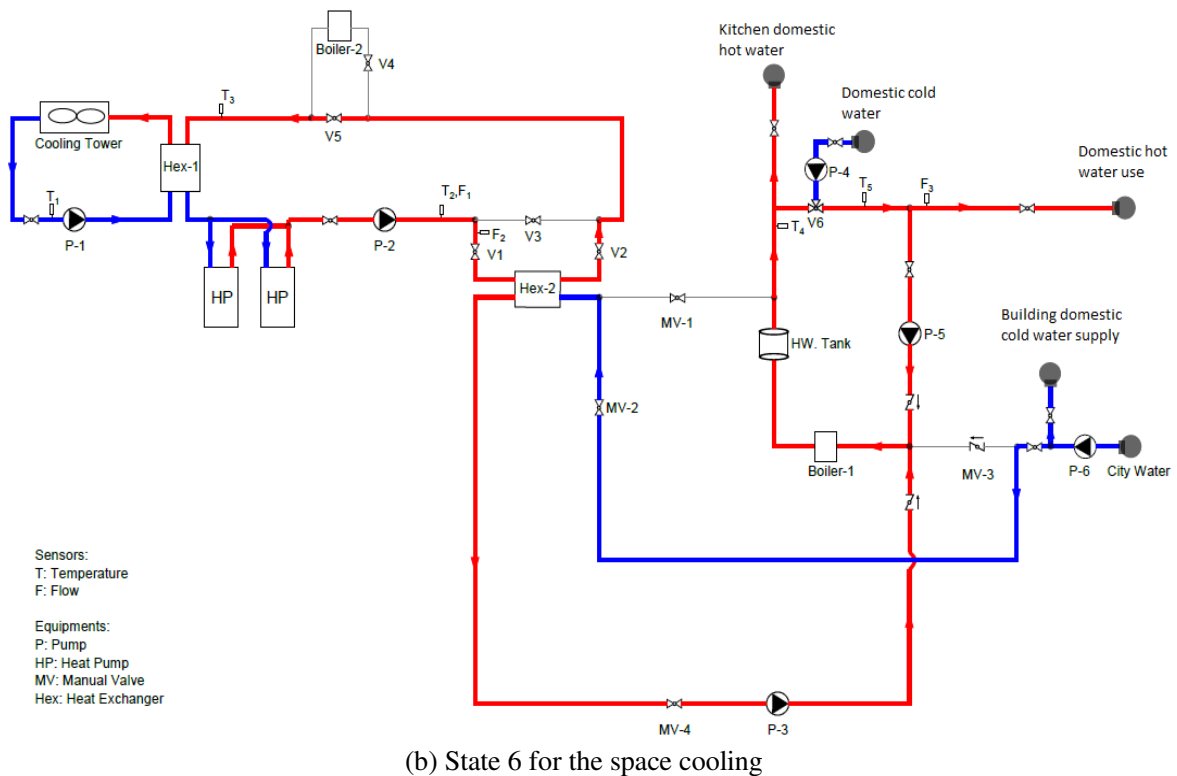
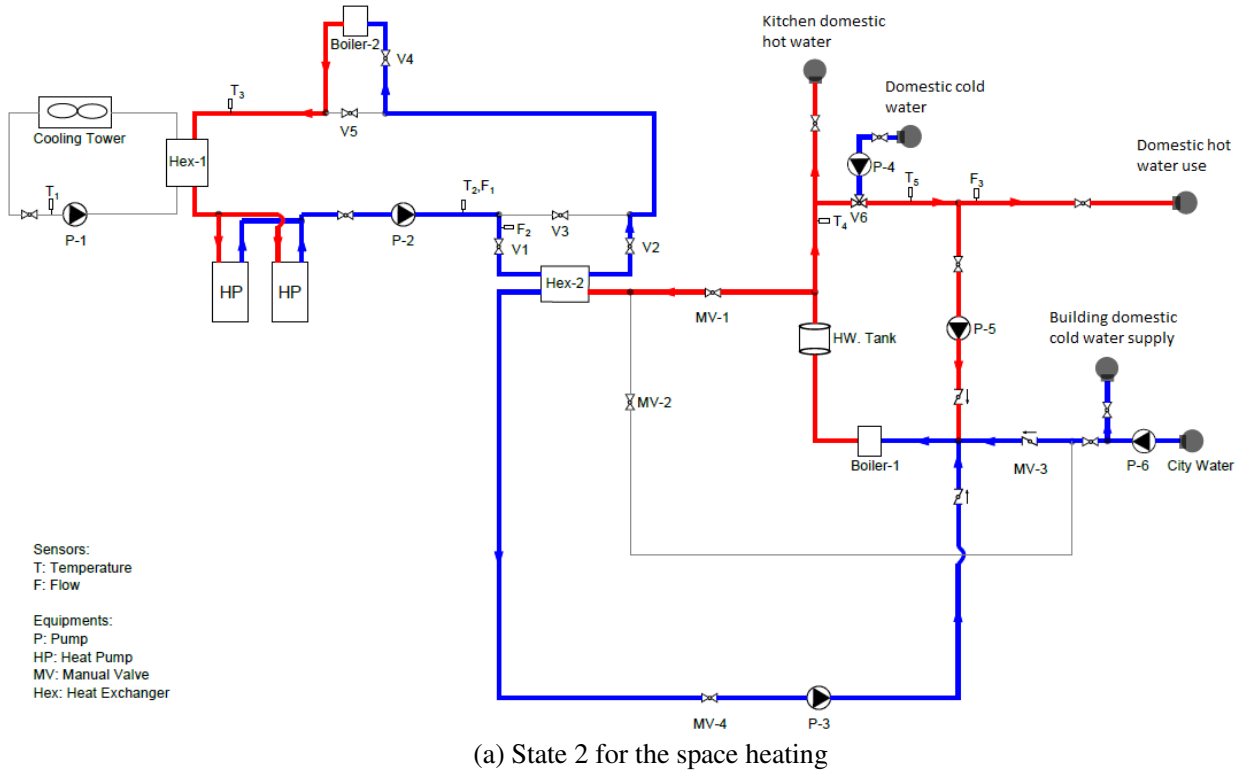


Figure 3. Two typical operating states of the proposed HR system

environment (State 6). This is anticipated to be the typical operating state for the space cooling since the DHW demand in the hotel mainly occurs in the early morning and late afternoon. For the rest of the day, the DHW demand is small but the cooling demand can be large. The flow direction in the connection loop at State 6 is shown in Figure 3b.

If it is hot and the DHW demand is too small (such as at night), it is not worth of running the heat recovery system. In that case, the DHW system and system conditioning system will operate independently at State 7.

The above analysis shows that the two subsystems operate jointly at State 2, 3, 5 and 6. However, the energy saving due to the heat recovery would only occur at State 5 and 6.

The transition between the states is achieved by employing a state machine (shown in Figure 4). In the state machine, temperatures, temperature differences and flow rates are used to indicate the different states of the whole system. The dead band and waiting time are employed to avoid short cycling.

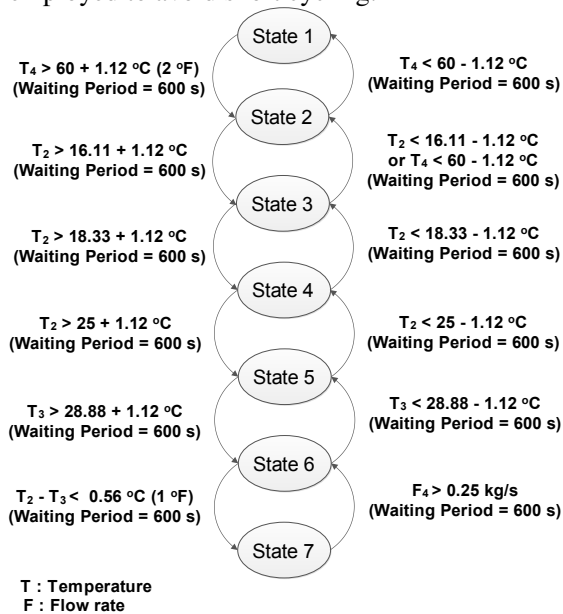


Figure 4. Supervisory control described by the state machine (the location of the temperature sensor and flow meter can be referred to Figure 3)

3 Evaluation

3.1 System Model

A commonly used method for the quantitative evaluation is to perform computer simulations (Gregor P. Henze, Clemens Felsmann *et al*, 2004; Hien, Poh *et al*, 2000). The widely used modeling tools in the building industry include EnergyPlus (Crawley, Lawrie *et al*, 2001) and TRNSYS (Klein, Duffie *et al*, 1976). However, it is difficult to use EnergyPlus to model the proposed heat recovery system because it does not support the unconventional system topology and tends

to highly idealize the control process (Huang and Zuo, 2014; Piette, Granderson *et al*, 2012; Wetter, 2009; Wetter, Zuo *et al*, 2011). TRNSYS is also not suitable for this case due to two limitations. First, TRNSYS is not effective in simulating such large system as the proposed design because it doesn't supply hierarchical modeling, which is essential for the debugging and model reuse (Wetter and Haugstetter, 2006); second, the pressure-driven flow distribution in the connection loop is hard to be modeled with TRNSYS. On one side, the flow direction in the connection loop varies by the operational states. On the other side, TRNSYS requires fixed and prescribed flow directions in the hydraulic system modeling (Kim, Zuo *et al*, 2013).

To overcome these challenges, we chose Modelica in the system modeling. The Modelica *Buildings* library (Wetter, Zuo *et al*, 2014; Wetter, Zuo *et al*, 2011) was used to build the physical system while *Modelica_StateGraph2* (Otter, Årzén *et al*, 2005) was employed to simulate the control system. The simulation platform is Dymola 2015 FD01.

Figure 6 shows the diagram of the top-level model for the whole system. It consists of five components: the DHW system, the connecting loop, the heat pump for the space conditioning, the condenser water loop, and the supervisor control system. Solid blue lines are pipes connecting the components and dashed lines are input or output signals for controls. We use the DHW system and the supervisor controller as examples to show the details of the Modelica models.

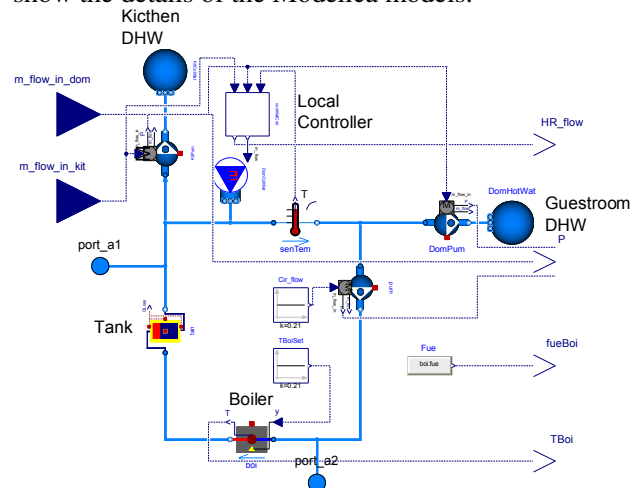


Figure 5. Diagram of the model for DHW system

Figure 5 is the model for the DHW system. The similarity between the system schematic (Figure 1) and Modelica models (Figure 5) allows a quick identification of modeling error. The DHW system model consists of physical equipment, such as a boiler (Boiler-1), a tank, and pumps, and the local controller for the temperature of DHW supplied to the guestrooms. This controller is committed to provide a 43.3°C hot water to the guestroom by mixing 60°C hot water from the boiler with the city water. The input of the DHW system model is the DHW demand for the

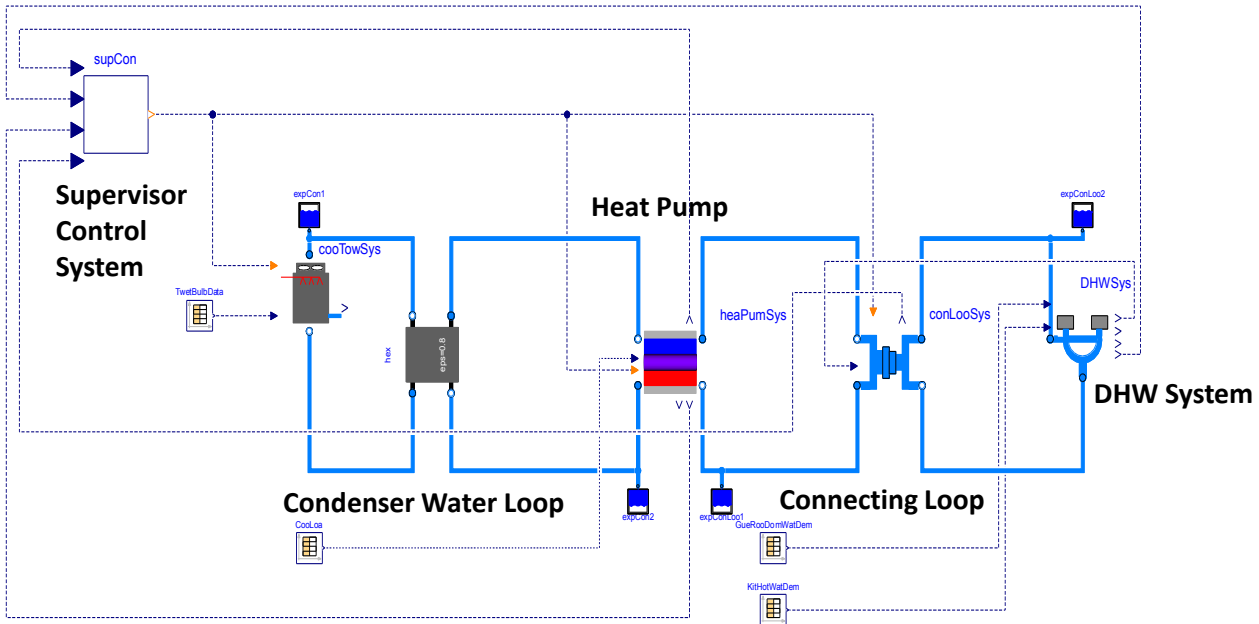


Figure 6. Diagram of the top-level model

kitchen and the guestroom while the output is the temperature of the DHW leaving Boiler-1 and the calculated recovering water flow rate as well as the energy consumption of Boiler-1 and so on.

Figure 7 shows the Modelica model for the supervisor control. The key part of this model is the state machine model, which consists of state (oval icon) and transition (bar icon) modules. The state modules were used to represent the seven states described in Figure 4. The input of the supervisor control model includes temperature of the condenser water entering and leaving the heat pump, the temperature of DHW leaving Boiler-1 and the heat recovering water flow rate. Its output is the state for the whole system.

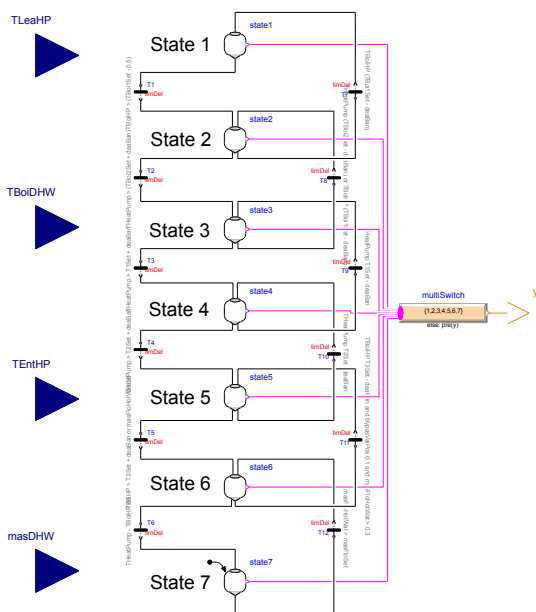


Figure 7. Diagram of the model for the supervisory control

3.2 Evaluation Setting

To evaluate the performance of the proposed heat recovery system, we simulated the system for one year period using a typical weather data. In addition, we studied the system for a typical cooling day and a special day.

For the annual simulation, there are three input variables: the weather data, the cooling load and the DHW demand. The weather data we used is the TMY (Typical Meteorological Year) file for the nearby Miami International Airport (U.S. Department of Energy). The cooling load is the heat that needs to be removed from the building. A negative sign of the cooling load means the heat is added into the building for the space heating (heating load). We used an empirical equation to calculate the cooling load according to the outdoor temperature:

$$\dot{Q} = \begin{cases} 4100 \left(\frac{T - 21}{11} + \frac{1}{2} \right) > -145 & 4100 \left(\frac{T - 21}{11} + \frac{1}{2} \right) \\ 4100 \left(\frac{T - 21}{11} + \frac{1}{2} \right) \leq -145 & -145 \end{cases} \quad (1)$$

where \dot{Q} is the cooling load (kW) and T is the outdoor air dry bulb temperature.

Furthermore, based on the engineering knowledge, we created a profile for the typical daily DHW demand in the hotel. The peak of DHW usage by the guestroom appears in the morning when guests get up and in late afternoon when guests come back from the beach. The DHW usage by the kitchen is mainly for preparing the lunch and dinner. The generated load and DHW profile is shown in Figure 8.

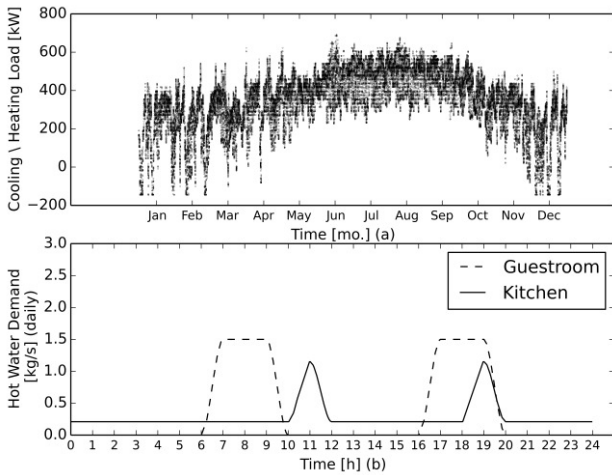


Figure 8. Cooling \ heating load and hot water usage profile for the annual simulation

For the typical cooling day, there would solely be cooling demand in the building. We used the same hourly DHW demand profile described in Figure 8 and a new hourly cooling load profile shown in Figure 9.

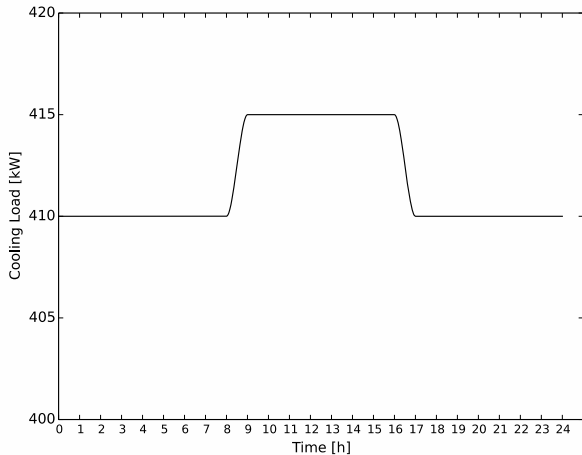


Figure 9. Cooling load for the typical cooling day simulation

For the special day (Feb 13), the heating and cooling was both needed as the day passed. We choose this day to evaluate the robustness of the supervisory control. We used the same hot water demand profile and use equation (1) to generate the load profile for cooling \ heating load data that is shown in Figure 10. According to the load, the operation mode of the space conditioning system should change from space cooling to space heating in the middle night. It then should turn back to space cooling in the morning.

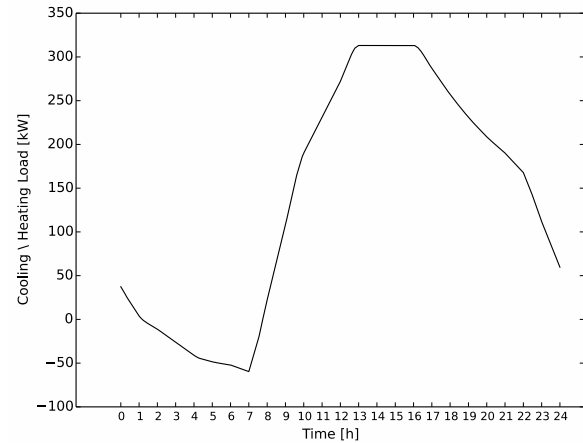


Figure 10. Cooling \ heating load for the special day simulation

3.3 Result

The result for the annual simulation is showed in Table 1 and Figure 11. The annual saving amount and saving ratio for Boiler-1 energy use is 411GJ and 19%, respectively.

Table 1. Annual simulation result

	Without heat recovery	With heat recovery
Boiler-1 annual energy consumption (GJ)	2,196	1,785
Boiler-1 annual energy saving ratio	N/A	19%

As expected, the system largely operated at State 5 to 7 and there was energy saving potential for Boiler-1 throughout the whole year (Figure 11). Most of the energy savings ranged from 20% to 30%. There were some days in the winter that the space heating was needed and the system ran at Sate 1 to 3. There are only a few hours that the system ran at State 1 when the cold weather and extreme large DHW demand happened at the same time.

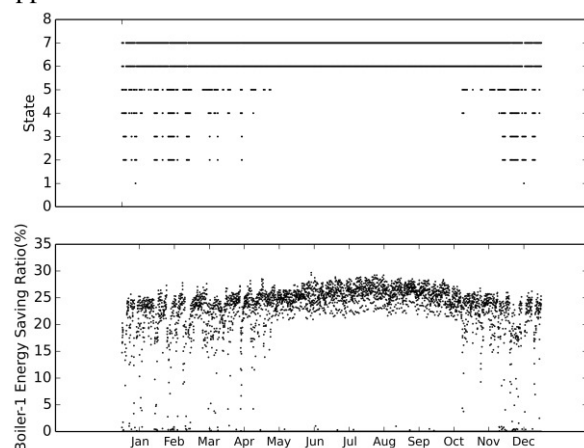


Figure 11. Annual simulation results

As shown in Figure 12 is the simulation result for the typical cooling day, the system mainly operated at States 6 and 7 depending on the DHW usage. If the DHW usage is sufficiently large, the heat recovery system ran at State 6 and saved about 25% energy for Boiler-1. Otherwise, the system ran at State 7 and there was no energy saving.

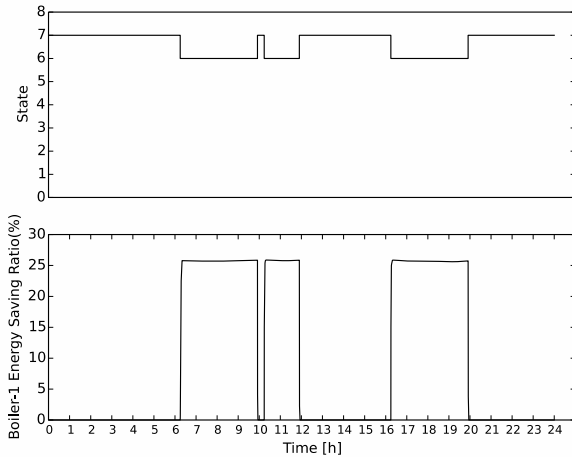


Figure 12. Simulation results for the typical cooling day

As shown in Figure 13, the system state in the special day changed from State 7 to State 2 in the early morning to switch from the space cooling to the space heating. It then switched back to the space cooling in the late morning. When the cooling load was below 200kW or there was only a little DHW demand, there was no energy savings taking place. However, when the cooling load rose above 200kW and there was DHW demand, Boiler-1 energy savings ratio spiked to just fewer than 25%. Then when the guestroom DHW demand was high between four to eight o'clock in the afternoon the boiler energy saving ratio stayed between 20% and 25%. The fall to 20% at the end of the time period could be attributed to additional DHW need for the kitchen and the lack of waste heat being transferred to warm up the 2.7kg/s of city water during the winter.

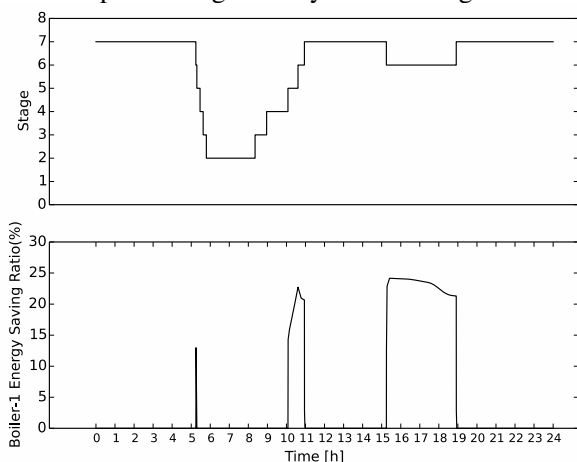


Figure 13. Simulation results for the special day

4 Conclusion

Based on the above analysis, we can find that,

- 1) the proposed heat recovery system can bring up to around 30% energy savings by the DHW boiler;
- 2) the special day simulation result showed that the proposed control system was able to regulate the relatively complicated system operation.

As we mentioned in the introduce section, the heat recovery system would affect not only the energy use of the boilers but also that of pumps and the cooling towers. However, due to the lack of the performance data, we couldn't make a quantitative analysis regarding the impact of heat recovery system on the pumps and the cooling towers energy use. Besides the energy saving, the reduction in the water usage by the cooling towers was also not considered since the water system was excluded from current simulation scope. At the next stage of this study, we will perform a more comprehensive evaluation after identifying the missing performance data for other equipment and including the water system in the system modeling. Based on those results, we may make recommendations for the design and control of heat recovery systems of hotels in tropical climate.

This study shows that there are advantages to using Modelica in the modeling of the building system. On the other hand, it still is challenging in debugging the Modelica models for the building systems using current Modelica environment, e.g. Dymola, since only limited information is provided during the simulation.

Acknowledgement

This work emerged from the Annex 60 project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme. Annex 60 will develop and demonstrate new-generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and BIM standards.

References

- ASHRAE. ANSI/ASHRAE/IES Standard 169-2006 -- Weather Data for Building Design Standards, 2006.
- Drury B. Crawley, Linda K. Lawrie, Frederick C. Winkelmann, W.F. Buhl, Y. Joe Huang, Curtis O. Pedersen, Richard K. Strand, Richard J. Liesen, Daniel E. Fisher, Michael J. Witte, Jason Glazer. EnergyPlus: creating a new-generation building energy simulation program. *Energy and Buildings*, 33(4): 319-331, 2001.
- Gregor P. Henze, Clemens Felsmann, Gottfried Knabe. Evaluation of optimal control for active and passive building thermal storage. *International Journal of Thermal Sciences*, 43(2004): 173-183, 2004.
- Wong Nyuk Hien, Lam Khee Poh, Henry Feriadi. The use of performance-based simulation tools for building design

- and evaluation - a Singapore perspective. *Building and Environment*, 35(2000): 709-736, 2000.
- Sen Huang, Wangda Zuo. Optimization of the water-cooled chiller plant system operation. In *Proc. of ASHRAE/IBPSA-USA Building Simulation Conference*, Atlanta, GA, U.S.A., 2014.
- Donghun Kim, Wangda Zuo, James E. Braun, Michael Wetter. Comparisons of building system modeling approached for control system design. In *Proc. of the 13th Conference of IBPSA*, Chambéry, France, 2013.
- S. A. Klein, J. A. Duffie, W. A. Beckman. TRNSYS – A Transient Simulation Program. *ASHRAE Transactions*, 82(1): 623-633, 1976.
- Martin Otter, Karl-Erik Årzén, Isolde Dressler. StateGraph – A Modelica Library for Hierarchical State Machines. In *Proc. of the 4th International Modelica Conference*, Hamburg, Germany, 2005.
- Mary Ann Piette, Jessica Granderson, Michael Wetter, Sila Kiliccote. Responsive and Intelligent Building Information and Control for Low-Energy and Optimized Grid Integration. In *Proc. of ACEEE 2012 Summer Study on Energy Efficiency in Buildings*, Pacific Grove, CA, U.S.A., 2012.
- U.S. Department of Energy. Buildings Energy Data Book. Retrieved May 15, 2014, from <https://catalog.data.gov/dataset/buildings-energy-data-book>.
- U.S. Department of Energy. EnergyPlus weather data. Retrieved May 18, 2015, from http://apps1.eere.energy.gov/buildings/energyplus/cfm/weather_data3.cfm/region=4_north_and_central_america_wmo_region_4/country=1_usa/cname=USA.
- U.S. Environmental Protection Agency ENERGY STAR Building Upgrade Manual, 2008.
- Michael Wetter. Modelica-based Modeling and Simulation to Support Research and Development in Building Energy and Control Systems. *Journal of Building Performance Simulation*, 2(2): 143-161, 2009.
- Michael Wetter, Christoph Haugstetter. MODELICA versus TRNSYS - a comparison between an equation-based and procedural modeling language for building energy simulation. In *Proc. of the 2nd National IBPSA-USA Conference*, Cambridge, MA, U.S.A., 2006.
- Michael Wetter, Wangda Zuo, Thierry Noudui, Xiufeng Pang. Modelica Buildings library. *Journal of Building Performance Simulation*, 7(4): 253-270, 2014.
- Michael Wetter, Wangda Zuo, Thierry Stephane Noudui. Recent Developments of the Modelica "Buildings" Library for Building Energy and Control Systems. In *Proc. of the 8th International Modelica Conference*, Dresden, Germany, 2011.

Presentation, Validation and Application of the *DistrictHeating* Modelica Library

Loïc Giraud, Roland Bavière, Mathieu Vallée, Cédric Paulus

Univ. Grenoble Alpes, INES, F-73375 Le Bourget du Lac, France
CEA, LITEN, 17, Rue des Martyrs, F-38054 Grenoble, France, roland.baviere@cea.fr

Abstract

District heating systems are a relevant solution for reducing CO₂ emissions, especially in dense areas with older buildings. However, due to the heavy investment costs, there is a great interest in simulation and software solutions to reduce distribution losses, limit the overuse of peak generators and optimize the use of storage capacities. In this paper, we describe how we designed, validated and used a library of fast, precise and robust components for district heating systems. Among other results, we could reduce the number of equations in some components by a factor of 40 and demonstrate more than 10% reduction in heat losses on a sample application.

Keywords: district heating, physical modeling, dynamic simulation, supply temperature optimization

1 Introduction

In French urban area, residential buildings currently account for 60% of the total energy consumption. This sector is also responsible for a large amount of carbon dioxide emissions. Most of this consumption is due to space heating and domestic hot water production. Following the recommendations of the “Grenelle Environment Round Table”, France is tied up to divide by four all emissions of greenhouse gases by 2050. New energy solutions must therefore be searched for the building sector. Generalizing standards of low consumption in new housings (e.g. RT2012 building code) as well as setting up incentives for building retrofitting can only be considered as long term measures since the renewal rate of existing buildings is limited to 1 %. On the other hand, district heating systems may already play a role since they are generally well established in dense urban area. Such system may massively increase the share of renewable and recovery energies, especially in urban areas where the use of decentralized systems is problematic or impossible. These context elements explain why France is currently experiencing a new age of development for district heating networks.

In France, large well-established district heating systems are continuing their extension while many

small networks are being built. However, due to the heavy investment costs of such systems, there is a great interest in simulation and energy planning software solutions leading to the reduction of distribution losses, limiting the overuse of peak generators and optimizing the use of centralized and decentralized storage capacities. In the Modelica community, previous work have been investigating issues related to short-term production planning in district heating networks (Velut *et al*, 2014). Our research group is currently involved in several research programs devoted to the definition of optimal operation of district heating systems. In this context we have been working on the development of computationally efficient and accurate dynamic simulation capabilities in order to propose and evaluate advanced control strategies.

The beginning of the present research program in January 2014 was devoted to the selection of an appropriate simulation platform, for instance able to host flexible model development. We have carried comparative studies of various candidates and the details of this analysis can be found in (Giraud *et al*, 2014). This work has led us to the conclusion that the equation-based object-oriented language Modelica along with the simulation platform Dymola was the most adapted tool for our application. This has led us to develop a Modelica component model library that we named *DistrictHeating*.

The purpose of this paper is to present the *DistrictHeating* library and to show how accurate modelling of a district heating network can be used as a basis to develop efficient control strategies. Section 2 first describes the structure and some of the models contained in the *DistrictHeating* library. Since the experimental validation of such models is an important issue, the validation process applied for the pipe model is entirely described in section 3. In section 4, we describe the optimization of a temperature control strategy in a virtual district heating network, designed to reproduce the behavior of a small part of the district heating network in Grenoble, France. Consumers are simulated using actual heat load profiles observed in the Grenoble main district heating system. Two supply temperature control strategies, a standard and an

optimized strategy, are compared in order to show the potential of energy savings.

2 Library for district heating system modelling and simulation

The current version of the *DistrictHeating* library contains several models intended to provide solutions for dynamic simulation of district heating and cooling systems. The modelling scope of the library is limited to systems using liquid water as the heat carrier fluid. As a consequence, the applicability of the developed models is restricted to cases where the fluid can be considered slightly- or in-compressible and poorly- or non-expandable.

2.1 Compatibility

Our *DistrictHeating* library relies on many modelling solutions provided by the version 3.2.1 of the Modelica Standard Library. For instance the *FluidPort* connectors defined in the *Modelica.Fluid* library (Franke *et al.*, 2009) and relying on stream variables are used. The *DistrictHeating* library can thus handle the flow-reversals which can occur for instance in meshed networks or networks with multiple supply points. The *HeatPort* connectors of the *Modelica.Thermal* library are also used. The modelling of one-dimensional thermo-fluid flows in piping network within the *DistrictHeating* library is compliant with the *Modelica.Media* package. We have also chosen to base our work on the *Modelica.Fluid* library since it allows modeling fluids with multiple trace substances. This feature can for instance be used to dynamically track the influence area of various production plants on the same heat grid. The compatibility of our model developments with other already existing, well documented and open-source modelica libraries such as *Buildings* (Wetter *et al.*, 2014) was also an important design criterion.

Although the *DistrictHeating* library is limited to our internal use for now, we can envision a wider diffusion of some components under an open-source license in the future.

2.2 Structure of the *DistrictHeating* library

The *DistrictHeating* library is composed of several packages containing model solutions for various components of a district heating system such as pre-insulated pipe, pump, substation, heat generator... The most important packages of our library are described in the following sections.

2.3 Package *Fluid*

The *Fluid* package provides components to model one-dimensional fluid flow in networks (see Figure 1).

The models representing one-dimensional pipes are gathered in the *Pipes* sub-package. The piping network of a district heating system is generally composed of

two identical parts namely the supply and the return networks. The twin-pipe configuration¹ apart, each piping element is composed of a pipe and a surrounding cylindrical thermal insulation. In some cases, it is necessary to account for the internal heat exchange between the supply and the return networks. In all cases, heat losses towards the surrounding environment should be accounted for. To ease the data collection burden for end-users we have developed a model representing a pair of pre-insulated pipes (see Figure 2) relying on geometrical parameters and solid properties available from vendors. We have collected such models in the *PipeShopCatalogue* sub-package.

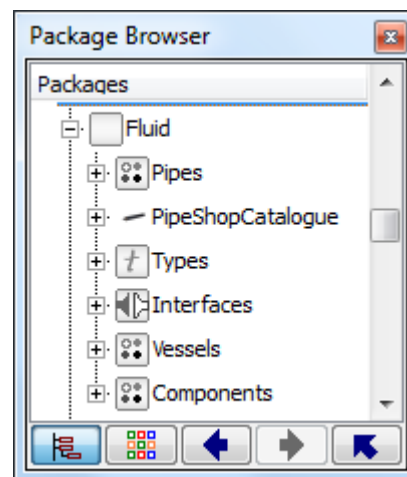


Figure 1. Structure of the *Fluid* package.

A model of tank with variable level is not useful for our application. We also found that head losses experienced by the fluid at inlet/outlet connections of a tank were generally negligible for our application. Following these two considerations, we have decided to introduce a *Vessels* sub-package in our library. This sub-package can be considered as a simplified version of the *Modelica.Fluid.Vessels* package.

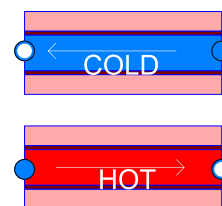


Figure 2. Icon representing a pair of pre-insulated pipes.

Some systems are built from n ($n \in \mathbb{N}$, $n > 1$) identical components (eg tube, pump ...) in parallel. For clarity reasons, let us only consider here the case of n tubes in parallel. When the operating conditions are similar for each tube, it is tempting to model the system relying on one elementary pipe affected with a weight of n . The weight feature is for instance available in the *DynamicPipe* model of the *Modelica.Fluid* library through the use of the *nParallel* parameter. However, implementing this

¹Configuration with two pipes in the same casing.

feature has led to the modification of 11 files within the *Modelica.Fluid.Pipes* sub-package. To ease the maintenance of our code, we have searched for a less intrusive mean of programming this feature. A *FlowDivider* and a *HeatDivider* model were developed for this purpose. These models are intended to be positioned at the interfaces of the system built from n identical components. Both models consist of a pair of ports introducing a division (respectively a multiplication) of the *flow* variables (mass flow-rate and heat flow-rate) in the positive (respectively negative) flow direction. Of course, the pressure drop and the accumulation terms for mass and energy are set to zero. These models are part of the *Components* sub-package.

2.3.1 Focus on the pipe model

Heat is convected at a velocity typically ranging from 0.05 m/s to approximately 2 m/s and over distances of a few to several tens of kilometers in a district heating network. As a consequence, a temperature change initiated at a production plant reaches far end-consumers with a delay that can exceed several hours. In order to limit heat losses throughout the distribution network, this delay must be correctly evaluated in order to overheat the network only when the thermal demand is high. Defining an optimal operation strategy for a district heating system therefore requires the use of a reliable pipe model correctly accounting for the temperature propagation dynamics. In order to define the best possible combination between accuracy and numerical performance we have developed two different numerical models to represent a district heating distribution pipe. Both models share common equations to express the momentum and mass balances across the pipe.

Several friction models ranging from linear relations to detailed laws accounting for the laminar-turbulent transition and the effect of roughness are available for end-users. However, a quadratic law fitted to data corresponding to nominal conditions and linearized in the laminar regime generally represents the best compromise between accuracy and numerical performance for district heating applications.

The following simplified equation derived from the first law of thermodynamics for open systems is used to express the 1-D fluid's energy conservation:

$$A \cdot \rho \cdot c_p \left(\frac{\partial T}{\partial t} + v \frac{\partial T}{\partial x} \right) = A \cdot \lambda \frac{\partial^2 T}{\partial x^2} + \dot{Q} \quad (1)$$

where A stands for the cross-sectional pipe area, ρ , c_p and λ respectively stand for the fluid density, specific heat capacity and thermal conductivity, T , v and \dot{Q} stand for the fluid temperature, velocity and the wall to fluid heat transfer-rate. As shown, in the validation section, the first term of the right hand side of Equation (1) can generally be ignored.

Equation (1) is a partial derivative equation (PDE) that requires the use of a numerical method to convert it in a form solvable by a computer program. We developed two pipe models based on two different numerical methods. These methods are respectively referred as the “element-” and the “node-“ method in the Danish scientific literature devoted to the modelling of temperature propagation within a district heating network (Benonysson, 1991; Gabrielaitiene *et al.*, 2008).

The ElementPipe model

In the first model, Equation (1) is spatially discretized using a finite volume method (Patankar, 1980). Relying on a collection of axially distributed non overlapping control volumes, Equation (1) is transformed into a system of ordinary differential equations that can be solved by a MODELICA tool. Each equation results from an integration of Equation (1) over an elementary control volume where the fluid temperature is assumed to be uniform.

The finite volume method is implemented using the *DynamicPipe* from the *Modelica.Fluid* library as a basis. The following modifications applied to the original *DynamicPipe* model are worth noting. Firstly, only one balance equation per pipe is considered for mass and momentum regardless of the number of control volumes used for the energy equation. Secondly, the discretization scheme for the convection term in the energy balance equation has been upgraded from an Upwind-Difference Scheme to the higher-order Quadratic Upstream Interpolation for Convective Kinematics (QUICK) scheme (Leonard, 1979). The QUICK scheme intends to limit artificial (also called numerical) diffusion. Thus, for a given accuracy level, larger mesh sizes can be used thereby improving the numerical efficiency of the model.

The NodeMethodPipe model

In the second model, Equation (1) is integrated along a fluid's particle path line following the method of characteristics (Wylie *et al.*, 1978). Along the path line, namely a characteristic curve, the PDE becomes an ordinary differential equation which can be natively integrated within a Modelica based computer program.

Following this method, it can be shown that the outlet temperature of a pipe can be inferred from a past inlet value according to the following equations (see (Giraud, 2015) for details):

$$T_{out}^*(t) = (T_{in}(t - \tau) - T_{ext})e^{-\frac{\tau}{\tau_t}} + T_{ext} \quad (2)$$

where $T_{in}(t - \tau)$ represents the pipe inlet temperature at the past instant $t - \tau$, τ is the transportation time, T_{ext} is the external temperature, τ_t is a heat loss characteristic time and $T_{out}^*(t)$ is the pipe outlet temperature obtained by neglecting the heat capacity of the tube.

τ is determined using (3) where L is the length of the pipe and v is the mean fluid velocity:

$$\int_{t-\tau}^t v(s) ds = L \quad (3)$$

$T_{out}^*(t)$ is then modified to account for the heat stored in the steel tube by assuming that the heat capacity of the whole tube is gathered at the outlet of the pipe and that the fluid and tube temperatures are equal:

$$m_a \frac{dT_{out}}{dt} = \dot{m} c_{p_f} (T_{out} - T_{out}^*) \quad (4)$$

where T_{out} is the fluid outlet temperature, m_a is the mass of the steel tube and \dot{m} and c_{p_f} respectively stand for the fluid mass flow-rate and the fluid specific heat capacity.

Equations (2), (3) and (4) have been implemented using a combination of the *delay(...)* and the *spatialDistribution(...)* operators (Modelica Association, 2014).

Table 1. Relative number of equations of the DAE system for a model composed of a pre-insulated district heating pipe and two boundary conditions

Model	Library	Rel. Nb. Eq.
<i>DynamicPipe</i>	<i>Modelica.Fluid</i>	100 %
<i>ElementPipe</i>	<i>DistrictHeating</i>	32.7 %
<i>NodeMethodPipe</i>	<i>DistrictHeating</i>	2.56 %

Table 1 compares the size of the system of Differential Algebraic Equations (DAE) for different pipe models. The number of meshes for the finite volume models (namely *DynamicPipe* and *ElementPipe*) is chosen such that the obtained numerical results are close to the results yielded by the model based on the method of characteristics (namely *NodeMethodPipe*) for a typical district heating transient. As can be seen from Table 1 the number of DAEs for the *ElementPipe* model is smaller by a factor of 3 to that of the *Modelica.Fluid DynamicPipe*. More importantly a reduction by a factor of 40 is observed for the *NodeMethodPipe*. Accordingly, significant computational costs savings are observed with this model when compared to the *DynamicPipe* or even to the *ElementPipe* models. In district heating system simulations, these numerical optimizations do not have any accuracy impact, as shown in the validation section (Figure 5). The *NodeMethodPipe* is thus used for the application shown in section 4.

2.4 Package Substation

The heat transported throughout a district heating network is delivered to consumers by the mean of a substation. The purpose of the *Substation* package is to provide generic model solutions to represent the thermohydraulic behavior of a district heating substation.

A substation is generally composed of a heat exchanger a control valve positioned on the primary side and a PI controller used to control the secondary outlet temperature (see Figure 3). Two distinct regimes must be considered to cover the operational conditions encountered in a district heating system. Firstly, in the thermal regime, the heat demand is satisfied and the primary mass flow-rate is entirely governed by the consumers' needs. On the other hand, in the hydraulic regime, the requested heat cannot be provided by the network. As a consequence, the control valve is fully opened and the primary mass flow-rate depends on the local pressure difference between the primary supply and return lines.

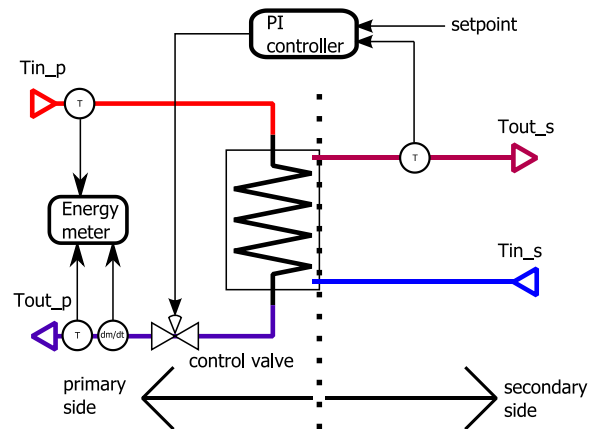


Figure 3. Schematic representation of a substation. T_{in_p} / T_{out_p} (resp. T_{in_s} / T_{out_s}) are the primary (resp. secondary) inlet / outlet temperatures.

We have developed several substation models in our library consisting of a control valve, a heat exchanger and a simplified controller. Details on this development work and on the validation procedure can be found in (Giraud *et al*, 2015). In the present paper, only the model representing the best compromise between accuracy and computational costs for our application will be described.

The purpose of a substation model is to cover simulation periods ranging from days to months. Consequently, the detailed dynamics of the PI controller can be ignored and the model may consider that the secondary outlet temperature always matches the set point value in the thermal regime. Correctly accounting for the heat exchanger behavior is a crucial part of the model. This part is based on the classical **LMTD** (Log Mean Temperature Difference) formulation (Shah *et al*, 2003):

$$Power = UA \cdot LMTD \quad (5)$$

where UA stands for the global heat exchanger thermal conductivity.

The **LMTD** term reads:

$$LMTD = \frac{\Delta T_1 - \Delta T_2}{\ln(\Delta T_1 / \Delta T_2)} \quad (6)$$

where $\Delta T_1 = Tin_p - Tout_s$ and $\Delta T_2 = Tout_p - Tin_s$ are the temperature differences at the two ends of the heat exchanger.

By considering that the solid/fluid heat transfers are similar for both sides of the heat exchanger and that the conduction in the solid part is negligible, UA can conveniently be expressed as:

$$UA = \frac{UA_{nom} \cdot [(\dot{m}p_{nom})^{-q} + (\dot{m}s_{nom})^{-q}]}{(\dot{m}p)^{-q} + (\dot{m}s)^{-q}} \quad (7)$$

where $\dot{m}p$ and $\dot{m}s$ respectively stand for the primary and the secondary mass flow-rates. The *nom* subscript indicates nominal conditions values and q is a user-defined parameter generally of the order of 0.7.

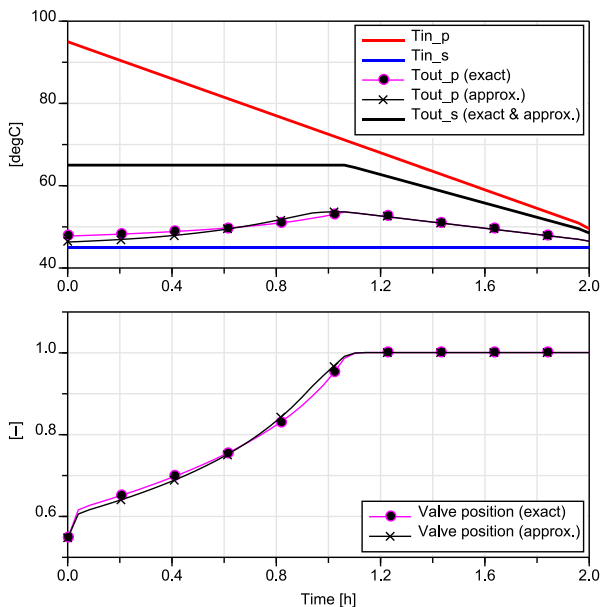


Figure 4. Evolutions of the temperatures (top) and valve position (bottom) for the exact and approximate substation models – for a decreasing network temperature.

Equations (5), (6) and (7) have been programmed in a Modelica model. When translated and solved by the DYMOLA FD01 2015 software, these equations lead to convergence and numerical stability issues, especially in the region where $|\Delta T_1 - \Delta T_2| / \Delta T_1 \ll 1$. As a workaround, the regularization method proposed in (Mattsson, 1997) was tested but the resulting model still suffered from numerical difficulties in the conditions of our application. Finally an alternative formulation, also linearized in the aforementioned region was implemented. In parallel, to improve numerical efficiency, we have searched for an approximate method relying on an explicit formulation that could be programmed in an *algorithm* section.

For the thermal regime, the heat exchanger behavior has been inferred from an explicit correlation between $\dot{m}p$ and the inputs of the model, see (Giraud *et al*, 2015) for details. In the hydraulic regime $\dot{m}p$ can be determined prior to the heat exchanger calculation. Equations (5), (6) and (7) can thus be solved explicitly in this regime.

Figure 4 compares the evolutions obtained with the “exact” and explicit (i.e. approximate) developed substation models. In the simulation scenario, the network temperature is progressively decreased while all other quantities are kept constant (power demand, network pressure difference, ...). Up to time 1.1 h, the heat demand is satisfied and the substation model runs in the thermal regime. In this regime, both models predict an increase in $Tout_p$ when Tin_p decreases. This is a coherent behavior for a heat exchanger with a UA coefficient that is only slightly sensitive to mass flow-rates variations. However, since this $Tout_p$ increase is limited, decreasing the supply temperature in a district heating network will generally lead to an overall heat losses improvement. In the hydraulic regime, $Tout_s$ decreases and the consumers’ heat demand cannot be fulfilled anymore. In summary, Figure 4 demonstrates a good qualitative behavior for our explicit substation model. We have also performed experimental validation relying on the analysis of temperature data recorded from instrumented district heating substations. Details of this validation results and procedures can be found in (Giraud *et al*, 2015).

2.5 Other Packages

The *DistrictHeating* Modelica library is also composed of many other packages providing solutions to model pumps, heat generator, stratified heat storage, conduction in multi-layers planar or cylindrical walls etc ... The thermophysical properties of the solid materials traditionally encountered in district heating and cooling systems are gathered in the *SolidMaterials* package.

3 Validation

In this section, we detail some of the experimental validation work that we have conducted for the pipe models described in section 2.3.1.

The accuracy of the different pipe models is assessed using the experimental data reported in (Ciuprinskas *et al*, 1999). The same experimental data have already been used in a similar validation work performed by (Gabrielaitiene *et al*, 2008). The measurements reported in (Ciuprinskas *et al*, 1999) have been obtained by triggering a temperature wave at a production plant of the Vilnius district heating network. The experiment has been conducted at the end of the heating season, when heat demand is low and network mass flow-rate is almost constant. Temperature measurements have been positioned at

both ends of an horizontal 470m in length pre-insulated pipe. Details on this case study and on the parameters that have been considered to build the corresponding model can be found in (Ciuprinskas *et al*, 1999; Gabrielaitiene *et al*, 2008).

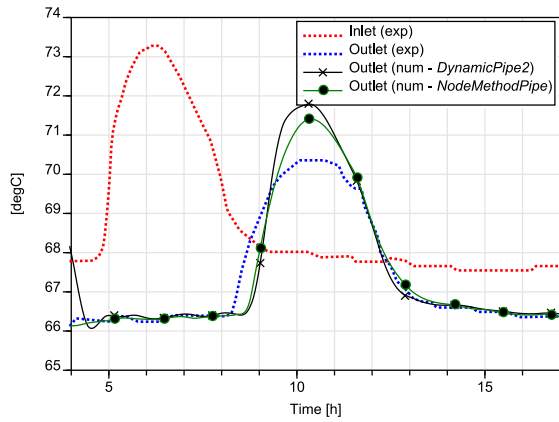


Figure 5. Numerical vs. experimental comparison of the temperature evolutions at both ends of an horizontal district heating pre-insulated pipe, 470m in length.

Figure 5 plots the experimental and the numerical results for two pipe models of our library, namely the *ElementPipe* and the *NodeMethodPipe* (see section 2.3.1 for details). This figure firstly shows that heat losses are correctly evaluated by the models since the temperature evolutions at the outlet are all equivalent. However, the numerical predictions slightly differ from the experimental evolutions concerning the peak temperature and the time at which the outlet temperature starts to rise. These errors exceed the experimental uncertainties (not shown here) reported in (Ciuprinskas *et al*, 1999). The same findings were also reported for comparable pipe models developed in a non-modelica environment (Gabrielaitiene *et al*, 2008).

In (Gabrielaitiene *et al*, 2008) it is postulated that the aforementioned numerical errors could originate from the fact that turbulent axial dispersion is significant in the conditions of the study yet this

phenomenon it is not accounted for by the models. Consequently, we have implemented a simplified version of the thermal diffusion and dispersion models proposed by (Drouin, 2010) in our *ElementPipe* model. This did not significantly improve our numerical predictions. We have also analyzed the potential impact of bends and other singularities on the temperature propagation dynamics by following the models proposed in (Park *et al*, 1971). Again, this did not improve our numerical predictions. We have also carried a sensitivity analysis on several uncertain experimental parameters (e.g. wall capacity, fluid/solid heat transfer coefficient, thermal dispersion coefficient ...). Within the considered variation ranges, it was not possible to significantly reduce the numerical errors.

Water tests on thermal stratification in a long horizontal pipe subject to transient inlet conditions were reported in (Tenchine *et al*, 2014). The purpose was to validate criteria for the prediction of occurrence and amplitude of thermal stratification in a simple horizontal pipe. The variation ranges regarding the experimental conditions explored in (Tenchine *et al*, 2014) are compatible with the experimental conditions of the present validation work in terms of pipe diameter, fluid mean velocity, amplitude of the inlet temperature transient but also Reynolds, Peclet, and Richardson dimensionless numbers. When applied to the district heating experimental conditions analyzed here, the criteria proposed in (Tenchine *et al*, 2014) suggest that the maximum cross-sectional temperature difference could reach 50 % of the inlet temperature wave amplitude. In other words, thermal stratification has probably occurred in the experiments reported in (Ciuprinskas *et al*, 1999). This would explain why the time at which the outlet temperature starts to rise is significantly under-predicted by the pipe models of our library which all rely on the assumption of flat velocity and temperature profiles. Since, the operational conditions leading to potential occurrence of thermal stratification in a district heating network (very low mass flow-rates and steep temperature changes) are

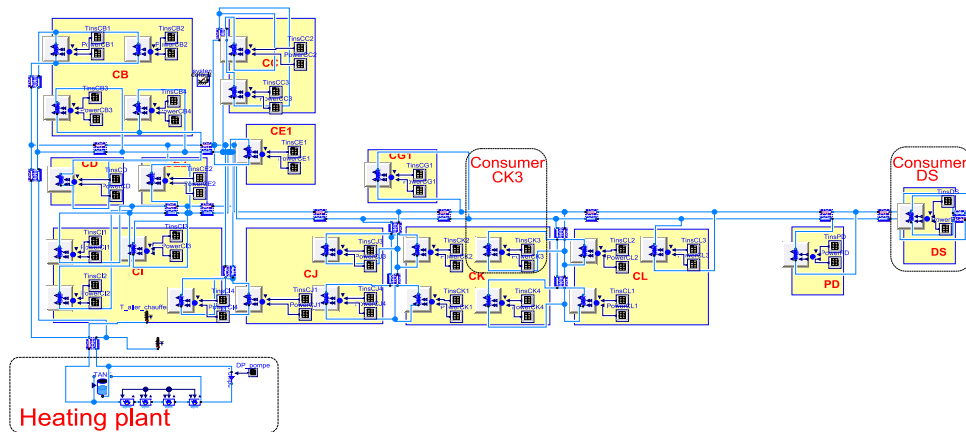


Figure 6. Layout of the sample district heating network, with 26 consumers and one heating plant.

very rare, we have decided to not include such a phenomenon in our pipe models. However, to complement the validation procedure exposed in the present paper, a dedicated experiment will be conducted in the Grenoble district heating network. More realistic experimental conditions will be targeted in order to exclude the potential occurrence of thermal stratification.

In summary, the numerical predictions produced by the available pipe models of the *DistrictHeating* library for a typical district heating transient are comparable to those obtained by other research group relying on non-Modelica tools (Gabrielaitiene *et al*, 2008). However, when compared to the experimental data reported in (Ciuprinskas *et al*, 1999), a slight numerical error is observed. By analysing the possible origins for this error, we have found that thermal stratification might have biased the temperature transport measurements performed in (Ciuprinskas *et al*, 1999). Further experimental work would be required to firmly conclude on this issue.

4 Application: optimized supply temperature

After presenting the library and its validation, we now present a concrete application of the library on a sample district heating network. In this section, we first give an overview of the realistic virtual district heating network we consider. We then describe the issue of choosing the supply temperature and the standard control law generally used in industry. We then show how the precise, fast and robust simulation obtained with the *DistrictHeating* library makes it possible to quickly obtain optimized supply temperatures results.

4.1 Overview of the virtual district heating network

Figure 6 depicts the virtual district heating network we use for this study. The mesh-free network layout considered in the virtual network originates from an extension project of the main district heating network in Grenoble, France. However, since the characteristics of the buildings in this new district are not yet available, we reconstructed heat load profiles based on historical data from other existing buildings in Grenoble.

In order to make this simulation as realistic as possible, we dimensioned the various virtual components carefully, taking into account the following constraints. Firstly, the buildings' profiles are chosen in order to respect the usual distribution of district heating clients in France, composed of 58 % households, 36% services and 6 % industries. Secondly, the substation models are dimensioned according to the local rules stating that the dimensioning load must be deliverable to consumers at a pressure difference of 1 bar and a heat exchanger

temperature difference of 110 K (i.e. $T_{in_p} - T_{in_s} = 110$ K). Finally, the pipes internal diameters are chosen within the range DN65 to DN350 in order to limit the maximal fluid velocity to 1.5 m/s. This leads to a maximum heat transportation time between the boiler and the far end consumer of 3 hours in the operating conditions investigated in the present work. The various piping elements of the virtual district network are modelled using the *NodeMethodPipe* Table 1.

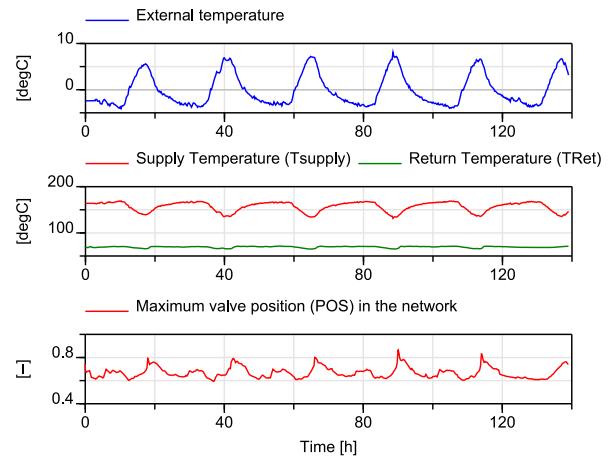


Figure 7. Evolution of the main variables over the simulation period (08/12/2013-13/12/2013).

The global heat losses coefficient of the network is adjusted to limit the relative losses to 10 % of the distributed energy during typical winter days.

The simulation period covers five consecutive days of December 2013 characterized by a cold yet sunny anticyclonic weather with daily temperature variations ranging between -4.1 °C and $+8.1$ °C. During this period, heat load patterns are regular, allowing for a particularly clear analysis.

4.2 Standard control of the supply temperature

In such a district heating network, correctly choosing the supply temperature at the heating plant is a good way of improving the efficiency of the whole system. On the one hand, the supply temperature must be high enough for substations to deliver the required power to customers. On the other hand, choosing higher supply temperatures increase heat losses during the transport, especially as transport times also tend to increase with higher temperatures. Perfectly optimizing the choice of the supply temperature would require taking into account the interdependencies between temperatures and transportation times for all consumers, as well as predicting the power requirements of all consumers over a few hours. In practice, even for a small network with a few dozen consumers, such a direct control proves impractical, both in terms of computation and in terms of investment costs for measurements.

As a consequence, the standard supply temperature control strategy uses an indirect solution, based on a static heating curve. With this strategy, the supply temperature is chosen as a linear function of the external temperature. The linear dependency between the supply and external temperatures can be adapted so that the maximal opening of the control valves never exceeds a given threshold, leaving a security margin for extreme cases.

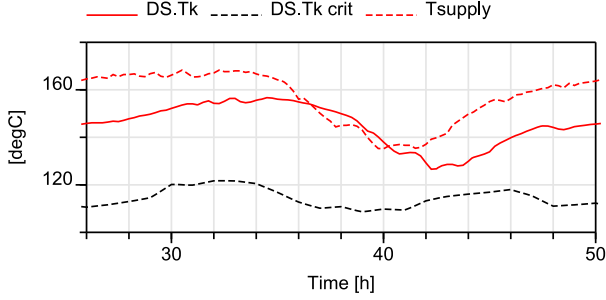


Figure 8. Supply temperature, network temperatures T_k and critical temperatures T_k^{crit} for consumer DS.

Using the virtual district heating network described above, we applied a standard supply control strategy, adapted so that the maximal opening of the control valves is limited to 85 % during the simulation period.

Figure 7 depicts the evolution of the main variables over the simulation period: the external temperature; the supply and return temperatures as well as the mass flow rate at the heating plant; and the maximal opening of control valves.

Since the choice of the supply temperature is only indirectly related to the heat loads at the consumer, and especially does not take into account transport times, we can observe that this standard control strategy leads to large variations of the opening of the control valves. This behavior is consistent with the observations in a real DH system.

Looking at the consumer side, we can use the substation models for a finer analysis. Figure 8 depicts the network temperatures at a consumer, named DS (red), together with the supply temperature (red dashed). One indicator we can compute thanks to the detailed substation model is the critical temperature (black), i.e., the minimal temperature required to be able to satisfy consumer demand. On these figures, we can see the actual network temperature is much higher than the critical temperature. For instance it can be about 150°C when only 120°C would be sufficient.

4.3 Optimizing the supply temperature

For each substation, we can compute the critical temperature, hereafter noted T_k^{crit} , which corresponds to the minimal network temperature allowing to serve the power required by consumer n° k . By taking into account transport times and heat losses, we can further compute the critical temperature at the heating plant according to the following equation:

$$T_{sup}^{crit}(t) = \max_{k,t'}(T_k^{crit}(t') + losses(\tau(t'))), \quad (8)$$

$$\forall t' \text{ such that } t = t' - \tau(t')$$

Choosing a supply temperature close to $T_{sup}^{crit}(t)$ ensures that demand at consumer side will be satisfied at all time, while minimizing heat losses. For a given set of heat load predictions, we can obtain all the data to compute $T_{sup}^{crit}(t)$ directly from the simulation results. However, since modifying the supply temperature also modifies mass flow rates, transport time, and thus network temperature, the selection of an optimized supply temperature is actually an iterative process.

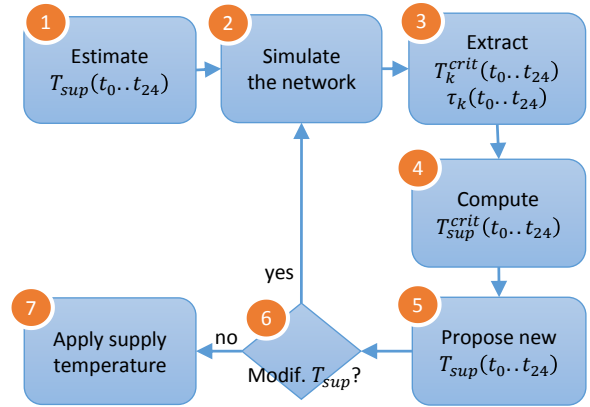


Figure 9. Computation of an optimized supply temperature, over a 24h prediction horizon ($t_0 \dots t_{24}$).

Figure 9 depicts the iterative process computing an optimized supply temperature:

1. Estimate an initial supply temperature schedule
2. Simulate the network using the Modelica *DistrictHeating* library
3. Extract critical temperatures and transport time for each consumers
4. Compute the aggregated critical temperature at the heating plant, taking into account individual critical temperatures and time delays (see Equation (8))
5. Choose a new supply temperature based on the aggregated critical temperature
6. Check if the supply temperature changed, and iterate to step 2 if necessary.
7. Once a solution is found, apply the supply temperature.

For our application, we implemented this algorithm in the Scilab scientific computing software (Scilab Enterprises, 2015), which communicates with Dymola for executing the Modelica simulation. This setting is close to the one adopted in previous work (Du *et al*, 2014), for optimizing dynamic hybrid energy systems.

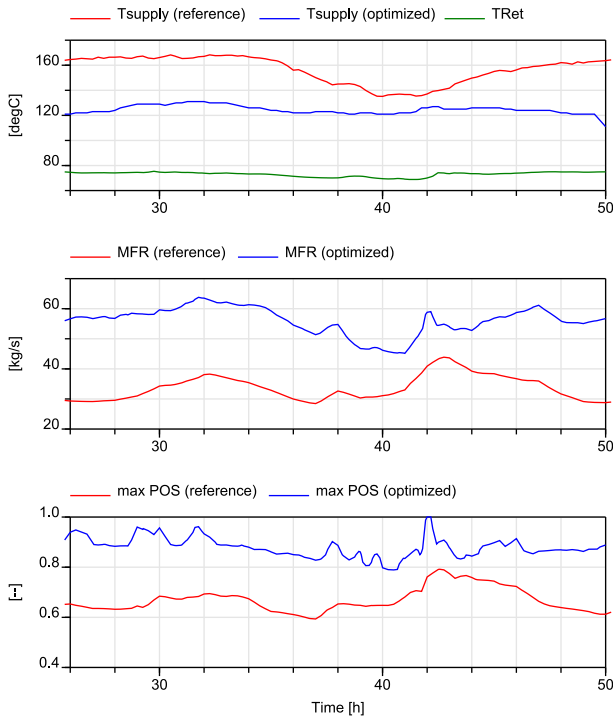


Figure 10. Main variables at the heating plant and maximal control valve opening, in the reference (red) and optimized (blue) cases. Return temperature at the heating plant (green) is similar in both cases.

4.4 Comparison between standard and optimized supply temperature

Figure 10 depicts the evolution of the main variables at the heating plant and the maximal control valve opening, both using the standard supply temperature (in red), and using the optimized supply temperature (in blue). In the upper figure we can see that the supply temperature can be reduced by up to 30°C , without reducing the ability to meet the consumers demand. In Figure 11, we can see that the produced thermal power is slightly reduced, because of a reduction of heat losses. Over the period we consider, the total energy savings amount to about 18% of the total heat losses.

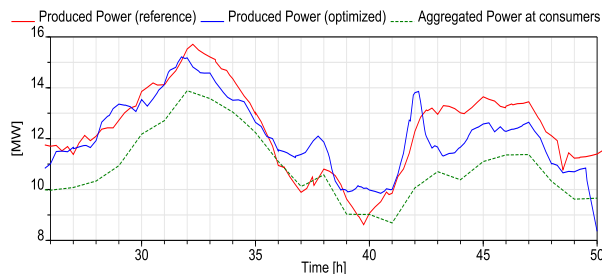


Figure 11. Produced thermal power in the reference (red) and optimized (blue) cases, and aggregated thermal power transferred to consumers (green).

Looking at the consumer side, Figure 12 clearly shows that the optimized network temperature (blue) is lower than using the standard supply temperature (red), and is closer to the critical temperatures (black

dashed). We can also observe that the control valves operate between 70% and 95% opening, and with more dynamic variations, indicating a finer control.

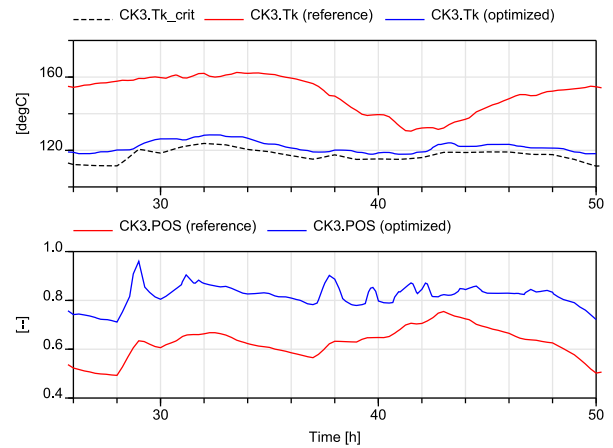


Figure 12. Results for consumer CK3 in the reference (red) and optimized (blue) cases: network temperatures (top), control valve position (bottom).

4.5 Discussion about heat load prediction

We can notice that heat load prediction for consumers plays an important role in the optimization of the supply temperature. In the work presented above, incorrect predictions could lead to lower network temperatures at the substations, and prevent from meeting the heat demand at some points in time. To mitigate this constraint, we can note the following points:

1. Since critical temperatures at consumers T_k^{crit} are close to linearly related to the heat load at each consumer, we can propagate prediction uncertainties to the choice of a supply temperature. For instance, if there is 10% uncertainty on the heat load prediction, increasing the optimized supply temperature by 10% of the typical temperature difference (approx. 1°C in most cases) would provide a good confidence to the network operator.
2. Even in case the network temperature is too low for meeting the heat demand during short periods of time, previous studies have shown that the thermal inertia of buildings ensure the inhabitants' comfort for at least a few hours (Kensby *et al*, 2015). In typical buildings, it is only after a few dozen hours that a reduction of heat power by 10% will lead to a reduction of internal temperature by 2°C .
3. To minimize prediction uncertainty, we can adopt a model-predictive control approach, in which the optimized supply temperature is updated on a regular basis with new predictions. Thanks to the rather slow dynamics of a district heating network, computing a new optimized supply temperature

every 15 minutes is enough to guarantee a fine-grained control.

5 Conclusion

In this paper, we describe how we designed, validated and used *DistrictHeating*, a library of fast, precise and robust Modelica components for district heating systems. In this library, we designed the components by iteratively improving standard components in order to reach a good balance between the precision and execution time of the simulation. We then validated the components by comparing their behavior with experiments on real district heating networks.

Based on the *DistrictHeating* library, we could design an application for optimizing the supply temperature of a sample district heating network. Using an optimized supply temperature allows reducing heat losses by 18% compared to a standard strategy. This optimized strategy relies on accurate prediction of the heat load for consumers.

As a further step, we will develop a model-predictive control approach for supply temperature optimization. In this approach, we can update heat load predictions regularly, and compute an optimized supply temperature for the next time slot accordingly. The improved simulation times of our library make it possible to adopt an update rate of a few minutes even for medium size systems (dozens of consumers), which is perfectly compatible with the slow dynamics of a district heating network. Using this approach, we plan to extend our study to a one-year simulation in order to estimate possible yearly savings on our test case.

Acknowledgements

The authors wish to thank Elise Le Goff, Nicolas Giraud and Philippe Clotot from CCIAG for their fruitful help in the realization of this study. We also would like to acknowledge the financial support of CCIAG for the joint research program and of ADEME for the PhD of Loïc GIRAUD.

References

- A. Benonysson, “Dynamic Modelling and Operational Optimization of District Heating Systems,” ISBN 87-88038-24-6. *PhD Thesis* - Technical University of Denmark, Lyngby, 1991
- K. Ciuprinskas and B. Narbutis, Experiments on heat losses from district heating pipelines. *Energetika* vol. 2 pp. 35–40, 1999
- M. Drouin, “Modélisation des écoulements turbulents anisothermes en milieu macroporeux par une approche de double filtrage,” *PhD Thesis* - Université de Toulouse, 2010 <http://www.theses.fr/2010INPT0066>
- W. Du, H. Garcia and C. Paredis, “An Optimization Framework for Dynamic Hybrid Energy Systems”, *Proc. of the 10-th International Modelica Conference*, Lund, Sweden, March 2014. [DOI: 10.3384/ECP14096767](https://doi.org/10.3384/ECP14096767)
- R. Franke, F. Casella, M. Otter, K. Proelss, M. Sielemann, and M. Wetter “Standardization of thermo-fluid modeling in Modelica.Fluid” in Francesco Casella, editor, *Proc. of the 7-th International Modelica Conference*, Como, Italy, September 2009. [DOI: 10.3384/ecp09430077](https://doi.org/10.3384/ecp09430077)
- I. Gabrielaitiene, B. Bøhm, and B. Sunden, “Evaluation of Approaches for Modeling Temperature Wave Propagation in District Heating Pipelines,” *Heat Transf. Eng.*, vol. 29, no. 1, pp. 45–56, 2008 [DOI: 10.1080/01457630701677130](https://doi.org/10.1080/01457630701677130)
- L. Giraud, R. Bavière and C. Paulus, “Modeling of Solar District Heating: A Comparison between TRNSYS and Modelica” *Proc. of EuroSun 2014*, Aix-les-Bains, France, 2014. [DOI: 10.18086/eurosun.2014.19.06](https://doi.org/10.18086/eurosun.2014.19.06)
- L. Giraud, R. Bavière, C. Paulus, M. Vallée and J.-F. Robin “Dynamic Modelling, Experimental Validation and Simulation of a Virtual District Heating Network” in *Proc. of the 28th Int. Conf. on Efficiency, Cost, Optimization, Simulation and Environmental Impact of Energy Systems (ECOS)*, Pau, France, 30th June – 3rd July 2015
- J. Kensby, A. Trüschel, and J.-O. Dalenbäck, “Potential of Residential Buildings as Thermal Energy Storage in District Heating Systems – Results from a Pilot Test.” *Applied Energy*, vol. 137 (January), pp. 773–781, 2015 [DOI: 10.1016/j.apenergy.2014.07.026](https://doi.org/10.1016/j.apenergy.2014.07.026).
- B. P. Leonard, “A stable and accurate convective modelling procedure based on quadratic upstream interpolation” *Comput. Methods Appl. Mech. Eng.*, vol. 19, pp. 59–98, 1979
- S. V. Mattsson, “On Modeling of Heat Exchanger in Modelica”, *Proc. of the 9th European Simulation Symposium, ESS'97*, Passau, Germany, Oct 19-23, 1997
- Modelica Association, *Modelica Language Specification version 3.3 Revision 1*, 2014
- C. M. Park and A. Gomezplata, “Axial dispersion in a tubular flow vessel with bends,” *Can. J. Chem. Eng.*, vol. 49, no. 2, pp. 202–206, 1971
- S. V. Patankar, *Numerical heat transfer and fluid flow*. CRC press, Taylor and Francis group, 1980
- Scilab Enterprises. *Scilab: Free and Open Source software for numerical computation*, 2015. Available from: <http://www.scilab.org>
- R.K. Shah and D.P. Sekulić, “Fundamentals of heat exchanger design” John Wiley & Sons, Inc., Hoboken, New Jersey , 2003.
- D. Tenchine and P. Gauthé, “Occurrence of thermal stratification in sodium cooled fast reactor piping,” *Nucl. Eng. Des.*, vol. 274, pp. 1–9, Jul. 2014
- S. Velut, P.-O. Larsson, J. Windahl, L. Saarinen, and K. Boman. “Short-Term Production Planning for District Heating Networks with JModelica.org.” *Proc. of the 10-th International Modelica Conference*, Lund, Sweden, March 2014. [DOI: 10.3384/ECP14096959](https://doi.org/10.3384/ECP14096959)
- M. Wetter, Wangda Zuo, Thierry S. Nouidui and Xiufeng Pang, Modelica Buildings library, *Journal of Building Performance Simulation*, 2014 Vol. 7, No. 4, 253–270, [DOI: 10.1080/19401493.2013.765506](https://doi.org/10.1080/19401493.2013.765506).
- E. B. Wylie and V.L. Streeter, “Fluid Transients”, McGraw-Hill Inc, New York, New York 1978

Multi-Mode DAE Systems with Varying Index

Sven Erik Mattsson¹, Martin Otter², Hilding Elmqvist¹

¹Dassault Systèmes, Sweden, {SvenErik.Mattsson, Hilding.Elmqvist}@3ds.com

²Institute of System Dynamics and Control, DLR, Germany, Martin.Otter@dlr.de

Abstract

This paper discusses an approach to handle multi-mode Differential Algebraic Equation (DAE) systems described by continuous-time state machines where mode-dependent state constraints are present. The goal is to perform static symbolic analysis and to generate efficient run-time code. This technique extends the class of multi-mode systems that can be handled by Modelica tools.

Keywords: Multi-mode, DAE, varying index, continuous-time state machine, variable structure system, symbolic transformation.

1 Introduction

1.1 Multi-Mode Systems

In (Elmqvist et al., 2014) a proposal was presented to extend the synchronous Modelica 3.3 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 and in particular acausal models. These new types of models are called “multi-mode systems”. An example of such kind of system is shown in Figure 1.

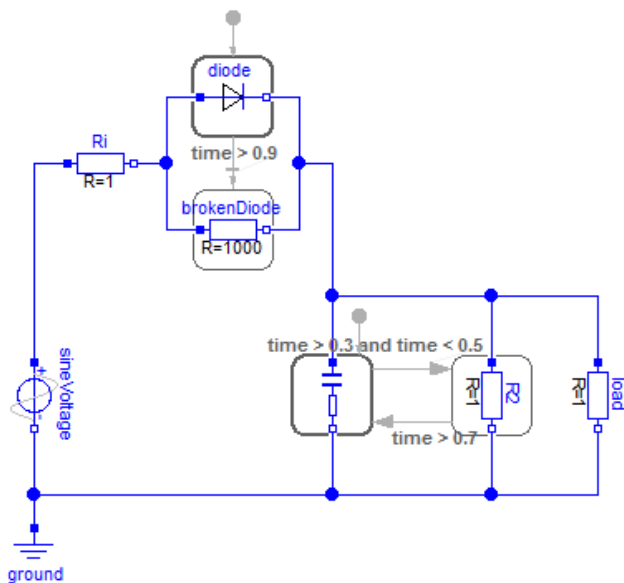


Figure 1. Circuit with two acausal state machines, from (Elmqvist et al., 2014).

Additionally, a method was developed to map connections to connectors of states in a particular way. The resulting equations can be processed basically by the *standard* symbolic algorithms supported by Modelica 3.2 tools. This approach already allowed handling a large class of useful variable structure systems with *dynamically changing number of continuous-time states*.

However, models could not 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. For example, the model in Figure 2 could not be handled. This circuit describes a capacitor C1 that is destroyed when the voltage becomes too large. The destroyed capacitor is modelled with a small resistor R1.

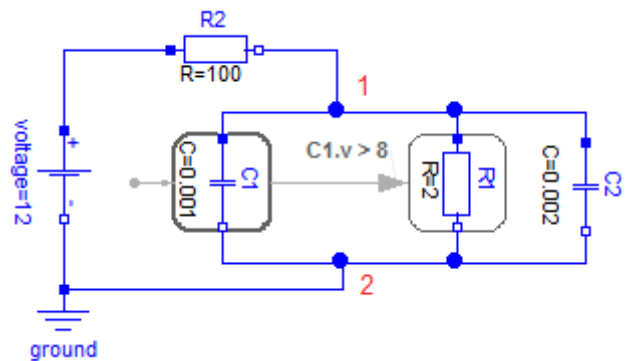


Figure 2. Circuit that could not be handled previously due to different state constraints in the different state machine states; slightly adapted from (Elmqvist et al., 2014).

The goal of this work is to extend the class of multi-mode DAE systems that can be simulated by Modelica tools and prototype the technique in Dymola (Dassault Systèmes, 2015).

In the EU research project RealSim, algorithms had been developed to symbolically process variable structure DAE systems and simulate the generated code (Mattsson et al., 2001). In particular, also a certain class of DAEs could be treated where Dirac impulses occur when switching the structure. These developments had not been incorporated into the release version of Dymola, because they had not been mature enough for a production software. Some ideas from the developments in the RealSim project are now being transferred to multi-mode systems.

1.2 Other Approaches

There are also other approaches to handle variable structure, varying index systems. For example (Zimmer, 2010) uses a run-time interpreter that processes the DAE equations at run-time, when the structure and/or the index is changing. The benefit is that a very large class of DAEs with varying index can be handled, at least in principal. The drawback is that the run-time efficiency is one or more orders of magnitude reduced with respect to an approach advocated by this article. Dynamically changing the structural analysis at run-time is also performed by (Höger, 2014). Describing variable structure systems with causal state machines is discussed by (Pepper et al., 2011).

(Benveniste et al., 2014) is tackling the problem from a fundamental point of view: The underlying, precise mathematical description is based on non-standard analysis for discrete-time and hybrid continuous-time/discrete-time multi-mode systems. This approach looks promising. However, it is not yet clear how to utilize this theory practically in a Modelica simulation environment.

2 Prerequisites

In this section several equations and properties are collected together that are prerequisites of the developed methodology, which is introduced in sections 3 and 4.

2.1 Connection equations

In (Elmqvist et al., 2014) it was shown how to map physical, acausal connections from components outside of a state machine to components on a state machine. An example is shown in Figure 2 where the components R2 and C2 are connected to the components R1 and C1 that form a *continuous-time state machine*.

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 m connectors $c_{e,j}$ outside of (that is external to) this state machine. Therefore, the following connect statements will be present (for simplicity it is assumed that exactly one of the external connectors, $c_{e,1}$, is connected to all the state machine connectors; if this would not be the case, one could always automatically re-arrange the connect statements):

```
connect(ce,1, c1)
...
connect(ce,1, cn)
connect(ce,1, ce,2)
...
connect(ce,1, ce,m)
```

These connect statements are replaced by the following equations, where i characterizes the active state of the state machine:

Connection equations

$$\begin{aligned} \mathbf{p} &= \{p_1, p_2, \dots, p_n\}; & \mathbf{p}_e &= \{p_{e,1}, p_{e,2}, \dots, p_{e,m}\}; \\ \mathbf{f} &= \{f_1, f_2, \dots, f_n\}; & \mathbf{f}_e &= \{f_{e,1}, f_{e,2}, \dots, f_{e,m}\}; \\ i &= \text{activeState}(); \\ p_{e,1} &= p_i; & & // \text{potential equations} \\ p_{e,1} &= p_{e,2:m} \\ 0 &= f_i + \sum_{j=1}^m f_{e,j} & & // \text{flow equation} \\ \text{for } r &\text{ in } 1:n-1 \\ & k = \text{mod}(i+r-1, n) + 1 \\ & 0 = \mathbf{h}_r(p_k, f_k) & & // \text{dummy equations} \\ \text{end for} \end{aligned} \quad (1)$$

Note, the for-loop generates $n-1$ dummy equations, $0 = \mathbf{h}_r(\dots)$. These dummy equations are only present in order that the equations of the non-active states form a regular system. The exact form of these equations is irrelevant because they are only used during symbolic analysis and are not present in the generated code. For more explanations, see (Elmqvist et al., 2014).

For connections of *two external* connectors to *two states* of a state machine, the connector equations of (1) simplify to:

Connection equations for the connection of two external connectors ce1, ce2 with two state machine connectors c1, c2:

$$\begin{aligned} & // \text{Equations for potential variables} \\ \text{ce1.p} &= \text{if activeState}(\text{state1}) \text{ then } c1.p \text{ else } c2.p; \\ \text{ce1.p} &= \text{ce2.p} \\ & // \text{Equation for flow variables} \\ 0 &= \text{ce1.f} + \text{ce2.f} + \\ & (\text{if activeState}(\text{state1}) \text{ then } c1.f \text{ else } c2.f); \\ & // \text{Dummy equation for not connected state} \\ 0 &= \text{if activeState}(\text{state1}) \text{ then} \\ & \quad \mathbf{h1}(\text{ce2.p}, \text{ce2.f}) \text{ else } \mathbf{h2}(c1.p, c1.f); \end{aligned} \quad (2)$$

The function `activeState(name)` is the Modelica 3.3 built-in function to inquire whether the instance *name* is the current active state of a state machine or not. Equations (2) are used in all following examples to map connections to state machines in to equations.

Input/output connections to states of a state machine can also be handled. Due to the known causality, this results in a much simpler approach as with acausal, physical connectors, and is not discussed in this paper.

2.2 Sinks and sources

The dummy equations (see last equation of (2)) have the drawback that they introduce algebraic loops between the states of a state machine and therefore make the analysis more difficult and the generated code less efficient (due to larger algebraic systems of

equations). These equations can be removed if either *one external potential* variable or *all external flow* variables are constants or known functions of time, in other words if the state machine states are connected to sink or source components. In such cases, equations (2) can be rewritten to:

Connection equations if *one external potential* variable $ce1.p$ is a constant or a known time function:

```
// Equations for potential variables
ce2.p := ce1.p
c1.p := if activeState(state1) then ce1.p else last(c1.p)
c2.p := if activeState(state2) then ce1.p else last(c2.p) (3)
```

```
// Equation for flow variables
0 = ce1.f + ce2.f +
  (if activeState(state1) then c1.f else c2.f);
```

or to

Connection equations if *all external flow* variables $ce1.f$, $ce2.f$ are constants or known time functions:

```
// Equations for potential variables
ce1.p = if activeState(state1) then c1.p else c2.p;
ce1.p = ce2.p

// Equation for flow variables (4)
c1.f := if activeState(state1) then ce1.f + ce2.f
      else last(c1.f)
c2.f := if activeState(state2) then ce1.f + ce2.f
      else last(c2.f)
```

Here $last(v)$ is a conceptual function (only used during symbolic analysis) to indicate the value of variable v from the last time instant where the corresponding state was active. For the symbolic analysis, $last(v)$ is a known value. The “:=” operator in the equations indicates the computational causality (= the left hand side is computed from the right hand side).

The proof of equations (3) is straightforward (a proof of equations (4) can be performed in a similar way): Start from (2) and recognize that the dummy equations of the not connected states, $h1(.)$ and $h2(.)$, can be arbitrarily selected, as long as the equations of the not connected states together with these dummy equations are *structurally consistent*¹. In (3) it is implicitly assumed that $state_i$ together with the rest of the system is *structurally consistent* if the state is active. If $state_i$ is not active, one can assume that keeping the causality of the connector (= identical to the case where the state is active) will still keep this non-active state together with its other dummy equations *structurally consistent*. In other words, under the assumption that $ce1.p$ is a constant or a known function of time, the last equation of (2) can be replaced by:

¹ A DAE is “structurally inconsistent”, if a unique solution cannot exist, or stated differently, if the *Pantelides algorithm* does not converge. (*Pantelides, 1988*) provides an algorithm to test for this property.

```
// Dummy equation for not connected state
0 = if activeState(state1) then
  c2.p - last(c2.p) else c1.p - last(c1.p); (5)
```

Rearranging the other equations of (2) together with (5) results in (3).

As a concrete example, let's analyze the circuit of Figure 2: At node 2 the state machine connectors $C1.n$, $R1.n$ and the external connectors $C2.n$, $voltage.n$, and $ground.p$ are connected together:

```
connect(ground.p, R1.n)
connect(ground.p, C1.n)
connect(ground.p, C2.n)
connect(ground.p, voltage.n) (6)
```

Since the potential of the ground component is given, $ground.p = 0$, one external potential variable of the connection set is a constant and therefore equations (3) can be utilized resulting in the following equations that are equivalent to (6):

Connection equations at node 2 of Figure 2:

```
// Equations for potential variables
R1.n.v = 0
C1.n.v = 0
C2.n.v = 0
voltage.n.v = 0 (7)
```

```
// Equation for flow variables
0 = ground.p.i + voltage.n.i + C2.i +
  (if activeState(R1) then R1.n.i else C1.n.i);
```

2.3 Differentiating dummy equations

When equations must be differentiated using a generalized form of the Pantelides algorithm (*Pantelides, 1988*), see section 3 and 4, dummy equations of non-connected states might need to be differentiated as well. For example assume that the equation for flow variables in (2) needs to be differentiated. When this equation is differentiated, the time derivatives of $ce1.f$, $ce2.f$, $c1.f$, $c2.f$ are introduced. This in turn means that also the dummy equation in (2) needs to be differentiated. Differentiating this dummy equation would utilize the newly introduced derivatives of the flow variables, but would also introduce *new derivatives* of the potential variables $c1.p$ and $c2.p$. This in turn might trigger other (unnecessary) differentiations.

We are rather free to select the dummy equations. They are only used to keep the equation sets of non-connected states *structurally consistent*. To avoid unnecessary state constraints of the dummy equations, and in turn unnecessary differentiations of equations, the dummy equations are actually defined in such a way that they provide a relationship between the actually occurring highest derivatives of the potential and flow variables.

For example, when the flow equation in (2) needs to be differentiated and the time derivatives of $ce1.f$, $ce2.f$,

c1.f, c2.f are introduced, then the dummy equation is changed to:

```
// Dummy equation for not connected state
0 = if activeState(state1) then
    h1(c2.p, der(c2.f)) else h2(c1.p, der(c1.f));
```

 (8)

Therefore, they provide a relationship between the differentiated flow variables and the non-differentiated potential variables.

2.4 Standard Pantelides and BLT algorithms

The *Pantelides algorithm* (Pantelides, 1988) is the key algorithm in this paper and will be generalized for multi-mode systems². It is summarized here for Modelica 3.2 DAEs, that is hybrid DAEs but without (discrete or continuous-time) state machines:

The flattened equations of a Modelica 3.2 model are described by the following equations:

Flattened equations of a Modelica 3.2 model

$$\begin{aligned} \text{(a) } \mathbf{w} &= \{\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t, \mathbf{m}, \mathbf{m}^-\} \\ \text{(b) } \mathbf{0} &= \mathbf{f}(\mathbf{w}, \text{relation}(\mathbf{w})) \\ \text{(c) } \mathbf{m} &= \mathbf{g}(\mathbf{w}, \text{relation}(\mathbf{w})) \end{aligned}$$
 (9)

where

- t The independent (real) variable
- $\mathbf{x}(t)$ Variables of type **Real**, appearing differentiated
- $\mathbf{y}(t)$ Variables of type **Real**, appearing not differentiated (= algebraic variables)
- $\mathbf{m}(t_{ev})$ Variables of type **discrete Real**, **Boolean**, **Integer**. They change their values only at event instants t_{ev} . At an event instant, \mathbf{m}^- is the value of \mathbf{m} at the previous event iteration at this time instant. During continuous integration, that is between events, \mathbf{m} is fixed (does not change) and $\mathbf{m}^- := \mathbf{m}$
- relation**(\mathbf{w}) All the relations in the model, for example $x_2 > y_5$. During continuous integration all relations are fixed (do not change).

If (directly or indirectly) constraints between variables \mathbf{x} are present, the Jacobian of (9b) with respect to the unknowns of type **Real** is singular:

$$\det\left(\begin{bmatrix} \frac{\partial f_i}{\partial \dot{x}_j} & \frac{\partial f_i}{\partial y_j} \end{bmatrix}\right) = 0$$
 (10)

This means that (9b) cannot be algebraically solved for the unknowns $\dot{\mathbf{x}}$ and \mathbf{y} . When using an explicit integration method to solve (9b) between events, these unknowns must be computed. Consequently, if (10)

holds, explicit integration methods cannot be used and initialization is problematic³.

The *Pantelides algorithm* solves this problem by differentiating singular subsets of equations. Since only equations are under consideration that are integrated, the starting point of the algorithm is equation (9b) where the discrete variables $\mathbf{m}, \mathbf{m}^-, \text{relation}(\mathbf{w})$ are kept constant (because they do not change during continuous integration) and therefore their dependencies are ignored:

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t)$$
 (11)

In particular this means that all **when**-clauses are removed and the dependency of the equations from conditions of **if**-clauses is ignored. The variable and equation structure of (11) is described in the following way:

- All variables appearing in (11) are collected in vector $\mathbf{v} = \{\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}\}$, $\mathbf{v} \in \mathbb{R}^{n_v}$ and (11) are $n_{f0} = n_x + n_y$ equations: $\mathbf{0} = \mathbf{f}(\mathbf{v}, t)$
- The variable association list \mathbf{V} is an **Integer** vector that defines if a variable v_j is the derivative of a variable i : $V_j = i$, if $\frac{dv_j}{dt} = v_i$. If no derivative of variable v_j is present, then $V_j = 0$ (i.e., these variables have the highest occurring derivatives).
- The equation association list \mathbf{F} is an **Integer** vector that defines if equation f_j is the derivative of an equation i : $F_j = i$, if $\frac{df_j}{dt} = f_i$. If no derivative of equation f_j is present, then $F_j = 0$ (i.e., these equations have the highest occurring derivatives).

After termination of the *Pantelides algorithm* the following two sets of equations are present:

$$0 = f_i(\mathbf{v}, t), \quad F_i = 0, \quad \frac{\partial f_i}{\partial v_j} \text{ structurally regular} \quad (12)$$

for $V_j = 0$,

and

$$0 = f_k(\mathbf{v}, t), \quad F_k > 0, \quad V_j > 0 \quad (13)$$

In other words, (12) are n_{f0} equations (the highest derivative equations) in n_{f0} unknowns (the highest derivative variables), and the highest derivative variables can be computed from the highest derivative equations (provided the Jacobian is not only structurally regular but also regular).

(13) are $n_f - n_{f0}$ equations describing the constraint equations between the potential states \mathbf{x} . They are also called invariants. The highest derivatives of the variables, v_j with $V_j = 0$, do not appear in these equations. These constraint equations can either be used to compute appropriate dummy derivatives with the Dummy Derivative method of (Mattsson and

² Most likely, the alternative formulation from (Pryce, 2001) can be used instead of the Pantelides algorithm as well.

³ see (Pantelides, 1988)

Söderlind, 1993), or these equations can be used to project the solution of (12) to the invariants (13) when the drift becomes too large.

For the new algorithm, the highest derivative equations (12) must be sorted to determine the execution order to compute the highest derivative unknowns. This includes determining the algebraic loops of this equation system. This is a standard algorithm for Modelica models and will be abbreviated as BLT (Block Lower Triangular) as it is usually done.

3 Basic Idea

In this section the basic idea to symbolically process multi-mode systems with varying state constraints (and therefore varying DAE index) is sketched at hand of the circuit of Figure 2. In the follow-up section 4 this idea is generalized and described in more detail.

Intuitively, when `activeState(C1)` is true, there are two capacitors in parallel, C1 and C2, and this results in a state constraint between the potential states C1.v and C2.v. The constraint equation must be differentiated, which means that the potential variables of the node 1 connection set, such as C2.p.v, *must be differentiated* as well.

When `activeState(R1)` is true, there is a capacitor C2 and a resistor R1 in parallel, and no state constraint is present. Therefore, the potential variables of the node 1 connection set, such as C2.p.v, *need not to be differentiated*. Since variables of external connectors to a state variable must be differentiated in one mode, and need not to be differentiated in the other mode, it is unlikely that it is possible to build one common equation system for all modes.

Such types of systems can be handled by an obvious *brute force method*: For every possible mode the equations of the complete system are generated for this particular mode and during simulation the model is switched between these equation sets. In the case of the circuit in Figure 2 there would be two equation sets. For small systems with only a few possible modes, this approach might be feasible. However, for large systems with many state machines and several states per state machine, the number of equation sets would be growing exponentially and the generated code would quickly become unmanageable. So this brute force method is not practical for the general case.

For this reason another approach is used that is inspired by (Mattsson et al., 2001). It is based on the property that differentiated equations contain the solution set of the non-differentiated equations. In the circuit of Figure 2 the potential variables of node 1 need to be differentiated when in state C1. We can accept this fact and use these differentiated potential variables also when in state R1. As a consequence, the potential equation $R1.p.v = C2.p.v$ is an invariant that must hold during simulation of its differentiated form. The benefit is that only one equation set can be

constructed for all modes. Lets' analyze this approach in more detail for the circuit in Figure 2:

The connection equations at node 2 are given by (7). The connection equations at node 1 are:

Connection equations at node 1 of Figure 2:

```
// Equations for potential variables
C2.p.v = if activeState(R1) then R1.p.v else C1.p.v
R2.n.v = C2.p.v

// Equation for flow variables
0 = C2.p.i + R2.n.i +
  (if activeState(R1) then R1.p.i else C1.p.i)      (14)

// Dummy equation for not connected state
0 = if activeState(R1) then
  h1(C1.p.v, C1.p.i) else h2(R1.p.v, R1.p.i);
```

Collecting all equations together and applying the *Pantelides algorithm* shows that no equation must be differentiated. The reason is that the if-clauses in (14) hide the state constraint between the two capacitors from the structural algorithm. BLT partitioning of the equations, taking into account the zeros in (7) and alias elimination, results in:

Sorted equations of Figure 2:

```
inputs = {C1.v, C2.v} // continuous-time states
```

```
R2.v = voltage.V-C2.v
R2.v = R2.R*voltage.i
```

```
algebraicLoop
```

```
unknowns = {R1.p.v, R1.p.i, C1.i}
```

```
// Local equations of R1
if activeState(R1) then
  R1.p.v = R1.R*R1.p.i
end if;
```

```
// Potential connections to the state machine
C2.v = if activeState(R1) then R1.p.v else C1.v
```

```
// Dummy equations for inactive states      (15)
0 = if activeState(R1) then
  h1(C1.v, C1.i) else h2(R1.p.v, R1.p.i)
end algebraicLoop
```

```
// Flow connection to the state machine
C2.i+voltage.i =
  -(if activeState(R1) then R1.p.i else C1.i)
C2.i = C2.C*der(C2.v)
```

```
if activeState(C1) then
  C1.i = C1.C*der(C1.v)
end if;
```

```
// Flow connection to the state machine
ground.p.i-C2.i-voltage.i =
  (if activeState(R1) then R1.p.i else -C1.i)
```

For BLT blocks with one variable and one equation, the equation with the variable to solve for is type-set in bold. In this example there is first a sequence of two such equations. After them there is an algebraic loop with 3 unknowns. The equations to use include local equations from the two states and external connection equations to them.

Every algebraic loop that contains connection equations to a state of a state machine must be analyzed whether it is (structurally) non-singular in all modes. For this, it is tried to make an assignment for every particular mode that can occur in the algebraic loop at hand. The algebraic loop in (15) gives rise to two modes: `activeState(R1)` is true or false. In both cases the relevant equations need to be extracted and the unknowns not present in this mode need to be removed:

Algebraic loop of (15) for `activeState(R1) == true`:

inputs = {C2.v} // continuous-time states

algebraicLoop

unknowns = {R1.p.v, R1.p.i}

R1.p.v = R1.R*R1.p.i (16)

// Potential connections to the state machine

C2.v = R1.p.v

end algebraicLoop

The algebraic loop in this mode consists of two equations in two unknowns. An assignment is possible, because `R1.p.v` can be assigned in the second equation and `R1.p.i` in the first equation. Therefore this set of equations is structurally regular.

Algebraic loop of (15) for `activeState(R1) == false`:

inputs = {C1.v, C2.v} // continuous-time states

algebraicLoop

unknowns = {C1.i}

(17)

// Potential connections to the state machine

C2.v = C1.v

end algebraicLoop

The algebraic loop in this mode consist of one equation in one unknown. Since the unknown `C1.i` does not appear in this equation, the algebraic loop is structurally singular.

The approach of Pantelides is to differentiate equations if the smallest possible set of equations has more equations as unknowns. In the new method we differentiate additionally the potential or flow connector equations from external connectors to connectors on a state of a continuous-time state machine if

these connector equations belong to an algebraic loop and in one mode this algebraic loops is (a)

singular and (b) the connector equation is present in this singular case.

In the above example, only equation

$$C2.v = \text{if } \text{activeState}(R1) \text{ then } R1.p.v \text{ else } C1.v \quad (18)$$

fulfills these requirements (it is a potential connector equation present in an algebraic loop, in mode `activeState(R1) == false` this loop is singular, and the equation is part of this singular loop). We differentiate this equation and adapt the corresponding dummy equation:

$$\begin{aligned} \text{der}(C2.v) &= \text{if } \text{activeState}(R1) \text{ then} \\ &\quad \text{der}(R1.p.v) \text{ else } \text{der}(C1.v) \\ 0 &= \text{if } \text{activeState}(R1) \text{ then} \\ &\quad \mathbf{h1}(\text{der}(C1.v), C1.i) \text{ else} \\ &\quad \mathbf{h2}(\text{der}(R1.p.v), R1.p.i) \end{aligned} \quad (19)$$

We take the original equations (15), remove (18) and its corresponding dummy equation and add (19). The (standard) *Pantelides algorithm* is performed on the resulting system. In this case the algorithm differentiates equations and variables. After termination, the highest derivative equations are structurally regular. Performing BLT partitioning on this structurally regular subset results in the following equations:

Sorted equations of highest derivative equations

inputs = {C1.v, C2.v, R1.p.v}

// continuous-time states + dummy states

R2.v = voltage.V - C2.v

R2.v = R2.R*voltage.i

R1.p.v = R1.R*R1.p.i

algebraicLoop

unknowns = { **der(C1.v)**, **der(C2.v)**, **der(R1.p.v)**,
C1.i, C2.i}

C1.i = C1.C***der(C1.v)**

C2.i = C2.C***der(C2.v)**

der(C2.v) = if activeState(R1) then der(R1.p.v)
else der(C1.v)

C2.i - R2.i = **-if activeState(R1) then R1.p.i**
else C1.i

0 = **if activeState(R1) then**
h1(der(C1.v), C1.i) else
h2(der(R1.p.v), R1.p.i)

end algebraicLoop

ground.p.i = C2.i - R2.i + (if activeState(R1) then
R1.p.i else C1.i)

Analyzing the newly occurring algebraic loop reveals that this loop is structurally regular in all modes (if this would not be the case, again potential or flow equations in the loop would be differentiated). Therefore, the overall algorithm can be terminated. The final equations have the property that the highest derivative equations are *structurally regular in all*

modes. The further processing and code generation is performed nearly in the same way as for Modelica 3.2 models. Especially, the dummy derivative method is applied (Mattsson and Söderlind, 1993) and a BLT of all highest derivative equations together with all discrete equations (9c) is performed as function of all unknowns. During code generation one has to additionally take into account that equations need to be deactivated when their corresponding state is not active.

Simulation results with the Dymola prototype are shown in the next figure:

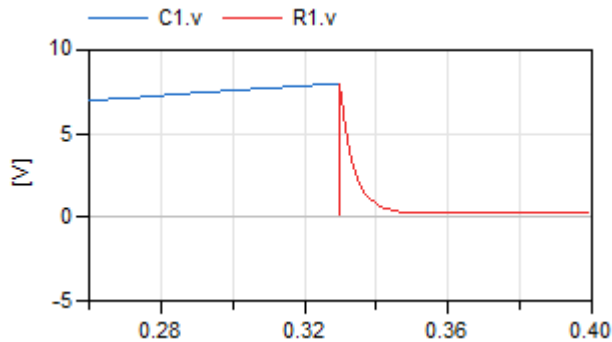


Figure 3. Simulation results of the circuit of Figure 2.

Since variable values of non-active states can have non meaningful values, Dymola only displays them in the time period, where the corresponding state is active. Therefore, C1.v is displayed only in the time range [0s, 0.33s] and R1.v is displayed only in the time range [0.33s, 0.6s].

4 Multi-Mode Pantelides Algorithm

The approach sketched in the previous section is more formally defined:

Starting point is a Modelica 3.2 model with one or more continuous-time state machines. As in section 2.4 all discrete equations and discrete variables are ignored during the following analysis. This also means that all transition conditions, such as $C1.v > 8$ in Figure 2, are ignored in this phase and the symbolic analysis is performed on equation (11). This equation is seen as a function of all (continuous-time) variables of type **Real** that appear in all states of all state machines and in all equations outside of all state machines.

Multi-Mode Pantelides Algorithm

1. Perform the standard *Pantelides algorithm* on (11) until convergence.
2. Perform BLT partitioning on the highest derivative equations (12) with respect to the highest derivative unknowns.
3. Analyze the algebraic loops detected in 2., that have at least one potential or flow connection equation (2) in the loop. For every such loop perform an assignment for every mode present in this loop (for the assignment ignore all variables and equations not

active in the particular mode).

4. *Stop*, if all algebraic loops in 3. are *structurally regular* for all modes. Otherwise, if an algebraic loop is *structurally singular* for at least one mode, stop the analysis of this loop after this first singular mode was found and *goto* 5.
5. For every loop in 4. that was found to be singular, the potential or flow connection equations that are (a) present in the respective loop and (b) give a structural singularity in the analyzed mode, need (conceptually) to be differentiated. This is indirectly achieved by introducing the differentiated variables of the respective connection equations in the variable association list.
6. Continue with the standard *Pantelides algorithm* by analyzing the highest derivative equations (without taking modes into consideration). After convergence is reached, *goto* 2.

The standard *Pantelides algorithm* differentiates the smallest possible set of equations that has more equations as unknowns. The generalization above additionally differentiates connection equations that are the result of connections to states of state machines, if algebraic loops become structurally singular when the corresponding state is active. After termination of the multi-mode Pantelides algorithm, (12) and (13) hold again. The new property is that (12) is structurally regular with respect to the highest derivative variables *in all modes!*

After step 2. it may happen that a connection equation is present outside of all algebraic loops and that this equation shall be solved for a potential or flow variable defined on one of the states. This is, for example, the case in the example of section 5.2:

```
// Flow connection to the state machine
L2.i = if activeState(diode.open) then          (21)
      diode.open.p.i else diode.closed.p.i;
```

This equation is structurally singular when state diode.open is active. Therefore, such an equation must also be differentiated in step 5.

The *multi-mode Pantelides algorithm* has a worst case complexity that grows exponentially with the number of possible modes which might be troublesome. However, in practice one can hope that this worst case complexity is usually not reached: Whenever state machines are present that influence each other not dynamically (say ideal diodes in an electrical circuit and friction components in the mechanical part of the model), then different algebraic loops will occur for the different state machines, the possible mode values in the loops will be different, and the analysis of the loops is decoupled.

One question is under which conditions the *multi-mode Pantelides algorithm* is converging (so stops after a finite number of iterations). For the standard

Pantelides algorithm this can be determined by replacing $\dot{\mathbf{x}}$ with \mathbf{x} in (11):

$$\mathbf{0} = \mathbf{f}(\mathbf{x}, \mathbf{x}, \mathbf{y}, t) \quad (22)$$

and performing an assignment for \mathbf{x} and \mathbf{y} . If this is possible, the algorithm converges. If not, the DAE (11) is *structurally inconsistent* and the algorithm does not converge. It is not yet clear how to generalize this property for the *multi-mode Pantelides algorithm*.

5 Examples

In this section further examples are shown that shall demonstrate the *multi-mode Pantelides algorithm* in different situations.

5.1 Varying index with inductors

The circuit in the next figure consists of two inductors in series, L1 and L2, where an over-current destroys L1 (the destroyed case is modeled with a large resistor).

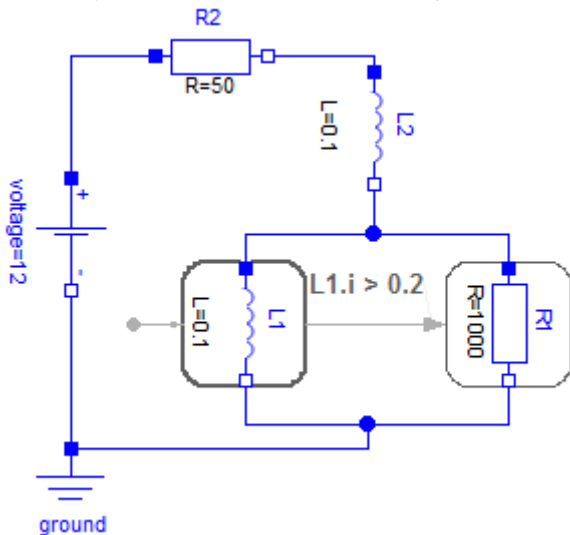


Figure 4. Inductors in series, where one of the inductors is destroyed when the current becomes too large.

When in state L1 the two inductors are in series and there is a constraint between the potential states L1.i and L2.i. When in state R1, this constraint is no longer present. The *multi-mode Pantelides algorithm* operates in a similar way as for the circuit in Figure 2. Simulation results are shown in the next figure:

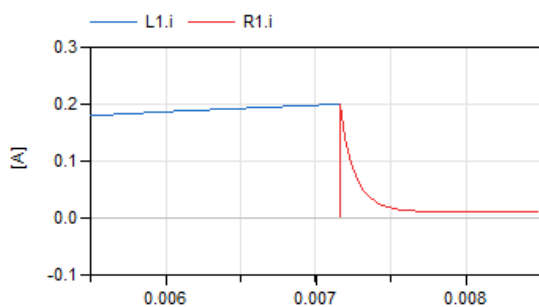


Figure 5. Simulation results of the circuit in Figure 4.

5.2 Varying index with inductor and diode

With continuous-time state machines it is possible to model ideal electrical switches, and in particular ideal diodes:

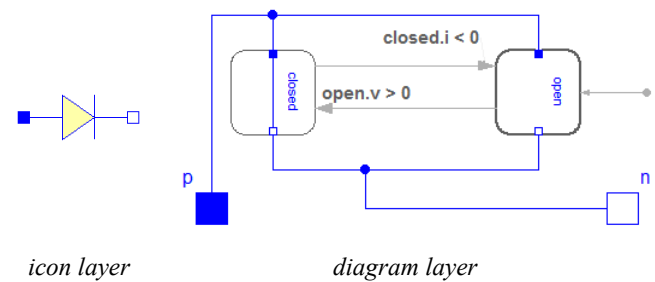


Figure 6. Ideal diode modelled with a continuous-time state machine.

The diode is modeled as a state machine where the first state is modeling a broken or open line and the second state is modeling an ideal line without resistance. For most situations there is no difference in using this diode model or the one from package Modelica (Modelica.Electrical.Analog.Ideal.IdealDiode) and setting $R_{on} = G_{off} = 0$. However, if varying state constraints occur this is different. Let us consider for example an inductor in series with a diode:

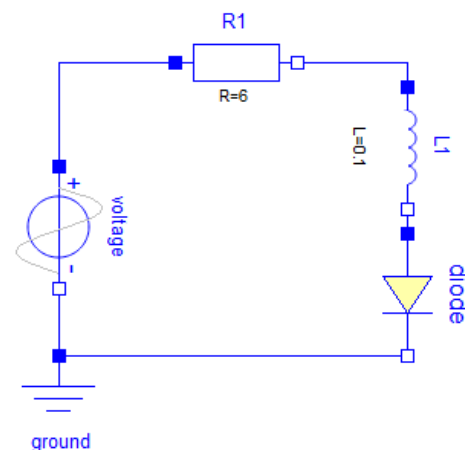


Figure 7. Inductor in series to an ideal diode model.

The current through the inductor, L1.i, is a state when the diode is in state “closed”. When the diode is in state “open”, the current through the diode is zero, which poses a state constraint forcing also the current through the diode, L1.i, to be zero, which means L1.i cannot be a state in that mode. Such circuits can now be handled with the *multi-mode Pantelides algorithm*, whereas using the ideal diode model from package Modelica would give a singular system during simulation.

Application of the standard *Pantelides algorithm* on the version with the ideal diode model of Figure 7 does not lead to differentiated equations. BLT does not lead to algebraic loops (provided the zero potential at the ground object is utilized). However, the sorted equations contain equation (21), as already discussed

in section 4. Since this equation is structurally singular when state diode.open is active, the differentiated connector variables, $\text{der}(\text{diode.open.p.i})$ and $\text{der}(\text{diode.closed.p.i})$ are newly introduced in the variable association list ($\text{der}(L1.i)$ is already present). In the next iteration of the algorithm an algebraic loop occurs which is structurally regular in both modes and the algorithm terminates.

5.3 Varying index with capacitor and diode

It is also possible to simulate the case of an ideal diode that is in parallel to a capacitor:

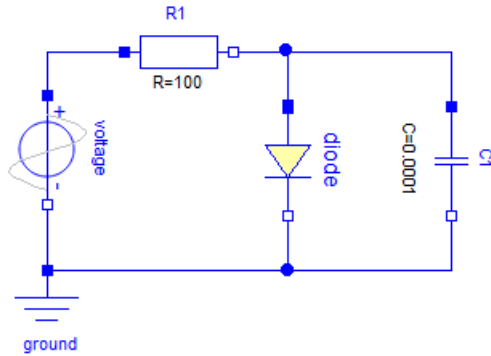


Figure 8. Capacitor in parallel to an ideal diode model.

When the diode is open, C1.v is a state, when it is closed, it is no state. The *multi-mode Pantelides algorithm* handles this system as well. It is interesting to compare simulations of this ideal diode model with the approximate ideal diode model of package Modelica:

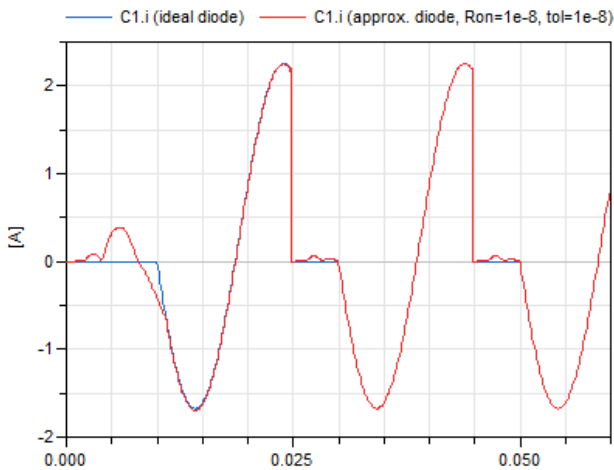


Figure 9. Simulation results of Figure 8.

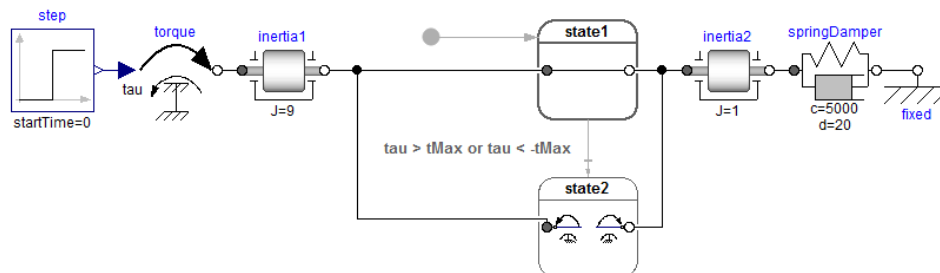


Figure 10. Shaft that breaks due to an overload torque $\tau > t_{Max}$ or $\tau < -t_{Max}$

Even for small values $R_{on} = G_{off} = 10^{-8}$ and strict relative error tolerances of 10^{-8} unphysical vibrations occur that are not present with the ideal diode model of Figure 6 giving the correct mathematical solution.

5.4 Varying index with breaking shaft

In Figure 10 a breaking shaft model is shown that could not be handled in (Elmqvist et al., 2014): In the beginning two inertias are rigidly connected together. When the absolute value of the cut-torque $\tau = \text{inertia2.flange_b.tau}$ becomes too large, the shaft breaks and two not-connected inertias remain. This is a case where three iterations of the *multi-mode Pantelides algorithm* are needed: In the first iteration the potential equations of the inertias (= flange angles) are differentiated, in a second iteration these differentiated equations are differentiated again, and in the third iteration it is recognized that the highest derivative equations are structurally regular for all modes. The Dymola prototype selects variables inertia1.phi and inertia1.w statically as states and then there are two conditional state selections for inertia2.phi and inertia2.w .

6 Limitations

The central result of this paper, the *multi-mode Pantelides algorithm*, was tested with several simple examples. However, much more tests especially with large models are needed. It might still be the case that improvements of the algorithm are needed. The following limitations are already known:

When using continuous-time state machines it is easy to model systems where Dirac impulses occur. For example, replacing the diode in Figure 8 by an electrical switch and closing this switch when the voltage drop is not zero, will result in a Dirac impulse. Simulation is usually successful. However, the “propagation” of impulses is not taken into account and therefore in many cases the simulation results will not be correct.

Another issue are the transition conditions: When they are functions of the state connector variables and these variables are differentiated, then the transition conditions might need to be differentiated as well. For example, friction can be modeled with the state machine of Figure 11 (the orange lines are mechanical connections that have angular velocity and not angle as potential variables).

The transition conditions from sliding to Stuck mode are the critical part: $w_{rel} > 0$ or $w_{rel} < 0$. When in Stuck mode, the constraint variable, w_{rel} , will be zero or close to zero and when switching from Stuck to Forward or Backward mode then small numerical errors will give different results, especially if dynamically coupled friction elements are present. It is well-known that for this switching direction the derivative of w_{rel} has also to be taken into account. It is not yet clear how to deduce this with an algorithm.

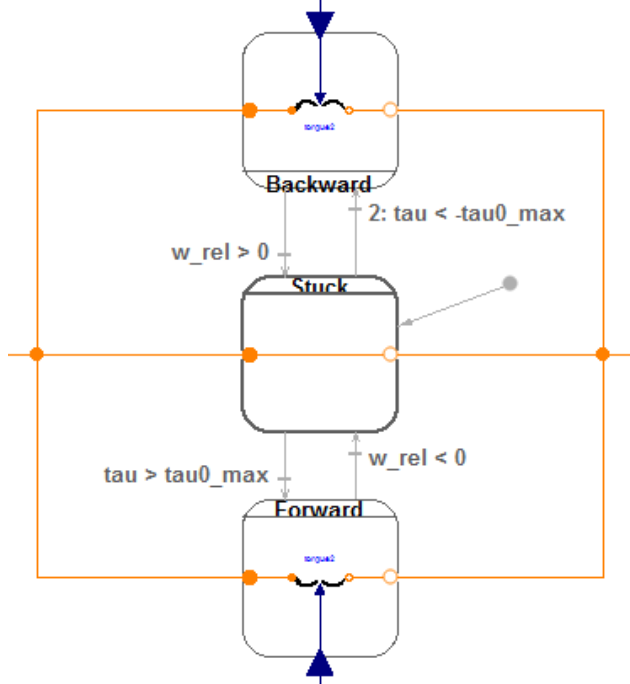


Figure 11. Model of a Coulomb friction element that cannot be handled with the approach of this paper.

7 Conclusions and outlook

In (Elmqvist *et al.*, 2014) a new approach was developed to define variable structure systems with varying number of continuous-time states in a convenient way with acausal continuous-time state machines. With a rather simple technique it was possible to symbolically analyze and simulate such systems. In the current paper the limitations of the previous approach have been reduced by generalizing the *Pantelides algorithm* for multi-mode systems. It is then possible to handle continuous-time state machines where state constraints can vary when switching to a new state. There are still unresolved issues and further development is needed before a robust and reliable solution becomes available for the user.

Acknowledgements

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

Additionally, inspiring discussions with Albert Benveniste and Benoit Caillaud about their approach to handle multi-mode systems are appreciated as well.

References

- Albert Benveniste, Timothy Bourke, Benoît Caillaud, Marc Pouzet (2014): **On the index of multi-mode DAE Systems (also called Hybrid DAE Systems)**. [Research Report] RR-8630, Inria; ENS. <hal-01084069>. Download: <https://hal.inria.fr/hal-01084069/document>
- Dassault Systèmes (2015): **Dymola 2016**. <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>
- Elmqvist H., Mattsson S.E., Otter M. (2014): **Modelica extensions for Multi-Mode DAE Systems**. Proceedings of the 10th International Modelica Conference, March 10-12, Lund, Sweden, pp. 183-193. Download: <http://www.ep.liu.se/ecp/096/019/ecp14096019.pdf>
- Höger C. (2014): **Dynamic Structural Analysis for DAEs**. Proceedings of the 2014 SCS Summer Simulation Multiconference. Download: http://dl.acm.org/ft_gateway.cfm?id=2685629&ftid=1511015&dwn=1&CFID=532067289&CFTOKEN=59766485
- 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.
- Mattsson S.E., Olsson H., Elmqvist H. (2001): **Methods and Algorithms for Varying Structure Hybrid DAE Simulation**. EC IST Project Realsim. Contract number: IST-1999-11979, Internal Report 2.2, Dynasim, Lund, Sweden.
- Modelica Association (2014): **Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.3, Revision 1**. June 11. Download: <https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf>
- Pantelides C. (1988): **The consistent initialization of differential-algebraic systems**. SIAM Journal of Scientific and Statistical Computing, 9(2), pp. 213-231.
- Pepper P., Mehlhase A., Höger C., Scholz L. (2011): **A Compositional Semantics for Modelica-style Variable-structure Modeling**. 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. ETH Zürich, Switzerland. Download: <http://www.ep.liu.se/ecp/056/006/ecp11056006.pdf>
- Pryce J.D. (2001): **A simple structural analysis method for DAEs**. BIT Numerical Mathematics, Vol. 41, No. 2, pp. 364-394.
- 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

Internalized State-Selection: Generation and Integration of Quasi-Linear Differential-Algebraic Equations

Christoph Höger¹ Andreas Steinbrecher²

¹Institute of Software Engineering and Theoretical Computer Science, TU Berlin, Germany,
christoph.hoeger@tu-berlin.de

²Department of Mathematics, TU Berlin, Germany, anst@math.tu-berlin.de

Abstract

In modeling and simulation of dynamical processes frequently higher index differential-algebraic equations (DAEs) arise. Since an attempt to solve higher-index DAEs directly yields several numerical problems, a regularization in combination with a robust and efficient integration is required. `QUALIDAES` is a DAE solver designed to make explicit use of such a regularization. It allows for the solution of over-determined quasi-linear DAEs of the form $M(x,t)\dot{x} = f(x,t)$, $0 = g(x,t)$. Such DAEs arise naturally if a quasi-linear DAE is regularized by augmentation with the set of its (hidden) constraints. General DAEs can be brought into the quasi-linear form. To this end, equations can be transformed into the specific input format expected by `QUALIDAES`. This transformation can be implemented in a functional style and yields a non-trivial result. Additionally it provides an on-the-fly solution for the occurrence of higher-order derivatives.

Keywords: Differential-Algebraic Equations, Quasi-Linear, Modelica, Translation, Regularization, Solver, `QUALIDAES`

1 Introduction

`MODELICA` is a language 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 Brenan et al. (1996); Griepentrog and März (1986); Hairer and Wanner (1996); Kunkel and Mehrmann (2006). On the other

hand, if a DAE does not contain any hidden constraint then its numerical treatment by use of implicit ordinary differential equation methods is not affected by instabilities. Furthermore, all constraints are preserved such that no drift-off effects arise in the numerical treatment.

Thus, a *regularization* or *remodeling* of the model equations resulting in an equivalent formulation with no hidden constraints is required to guarantee stable and robust numerical computations, see also Gear (1988); Hairer and Wanner (1996); Kunkel and Mehrmann (2006); Steinbrecher (2006).

The current state of the art in many modeling and simulation tools to deal with high index DAEs is to use some kind of analysis of the system 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 in Section 2 a different regularization approach for the remodeling of dynamical systems that uses the hidden constraints to construct an over-determined system regularization that can be solved using a specially adapted numerical integrator implemented in the software package `QUALIDAES` (`QUASI LINEAR DAE Solver`), see Section 3. This approach is developed for the numerical treatment of *quasi-linear DAEs* of the form

$$E(x,t)\dot{x} = k(x,t) \quad (1)$$

and has the great advantage that the problem of state selection can be moved into the numerical integrator such that it can be performed during the run-time of the simulation.

The software package `QUALIDAES` requires and exploits the quasi-linear structure of the model equations. If `QUALIDAES` is used in the `MODELICA`-framework then this requires a representation of the `MODELICA` model equation in quasi-linear form, as illustrated in Section 4.

2 Regularization Using Over-determined Formulations

In the following, we consider quasi-linear DAEs of the form (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 non-redundant. 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 $(x, t) \in \mathbb{R}^n \times \mathbb{I}$. Regularization approaches for high index DAEs like the Dummy Derivatives Approach from Mattsson and Söderlind (1993) or index reduction by Minimal Extension from Kunkel and Mehrmann (2004) consists of adding the hidden constraints to the model equations and the selection of certain differential components of the state x 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 selection of differential components can change during the numerical integration. Thus, if this state selection is performed outside the numerical integrator this often takes too long and is computational inefficient.

In the following, we will present a regularization of quasi-linear DAEs (1) of higher index, i.e., that contain hidden constraints. This regularization is based on an over-determined system formulation in order to overcome the difficulties in the numerical simulation. Certain analysis tools, like Pantelides' algorithm (Pantelides (1988)), the structural analysis by Pryce (Pryce (2001)), the analysis via the strangeness-index concept (Kunkel and Mehrmann (2006)), the algebraic procedure proposed in Steinbrecher (2006), a combined structural-algebraic approach proposed in Scholz and Steinbrecher (2013), or other, gives us the required information about the hidden constraints in the system.

These information consists mainly of the order of differentiation of (parts of) the DAE to determine the hidden constraints by algebraic manipulations of the equations and their derivatives. The minimal order of differentiation of (parts of) the DAE required for the determination of a certain (hidden) constraint is called the *level of the (hidden) constraint*. Furthermore, the maximal level v_c of existing hidden constraints is called *maximal constraint level* of the DAE. See Steinbrecher (2006). Let us denote the set of all constraints including the hidden constraints by

$$0 = h(x, t). \quad (2)$$

Adding the hidden constraints to the quasi-linear DAE

(1) leads to an over-determined DAE

$$E(x, t)\dot{x} = k(x, t), \quad (3a)$$

$$0 = h(x, t) \quad (3b)$$

consisting of a differential part (3a) and an algebraic part (3b). This over-determined formulation (3) then is equivalent to the original DAE (1) in the sense that both have the same solution set, i.e., for a given consistent initial value, the corresponding initial value problems have the same solution. Note that the leading matrix E not necessarily has to have full rank and 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 over-determined formulation (3) has the advantage that all constraints explicitly are stated, Therefore, for (3) no hidden constraints exist. A further advantage of the over-determined formulation (3) is the fact that it is not necessary to apply analytic manipulations for the determination of a square, regular system of DAEs. The proposed remodeling can be seen as regularization of the model equations. For more details on the regularization of quasi-linear DAEs we refer to Steinbrecher (2006).

Example 2.1 The Cartesian Pendulum: Let us con-

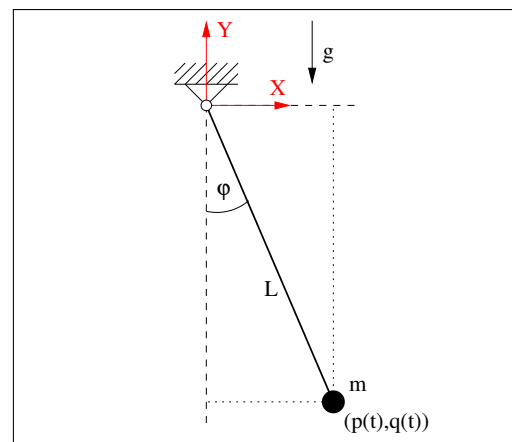


Figure 1. Topology of the Cartesian pendulum

sider the Cartesian pendulum, see Figure 1. We choose absolute coordinates p and q denoting the position of the mass m in the two dimensional space \mathbb{R}^2 for the description of the configuration of the pendulum. The equations of motion have the form

$$\dot{p} = v, \quad (4a)$$

$$\dot{q} = w, \quad (4b)$$

$$m\dot{v} = -2p\lambda, \quad (4c)$$

$$m\dot{w} = -mg - 2q\lambda, \quad (4d)$$

$$0 = p^2 + q^2 - L^2, \quad (4e)$$

where v and w denote the velocities of the mass point in X - and Y -direction while λ corresponds to the Lagrange

multiplier. The constraint (4e) is of level 0 since no differentiation of the DAE is necessary to determine this constraint. The hidden constraint of level 1 obtained after only one total differentiation of (4e) with respect to t and replacing \dot{x} and \dot{y} using (4a), (4b) is given by

$$0 = 2pv + 2qw. \quad (4f)$$

Furthermore, from the second total derivative of (4e) with respect to t and replacing \ddot{x} , \ddot{y} using the the first total derivative of (4a), (4b), and subsequent replacing of \dot{x} , \dot{y} , \dot{v} , and \dot{w} using (4a)-(4d) we get the hidden constraint of level 2 as

$$0 = 2v^2 + 2w^2 + 2p(-2p\lambda)/m + 2q(-mg - 2q\lambda)/m. \quad (4g)$$

A further differentiation of the DAE does not lead to further constraints. Consequently the minimal order of differentiation of (parts of) the DAE to determine all (hidden) constraints is two. Therefore, the model equations for the Cartesian pendulum is a set of DAEs of maximal constraint level $v_c = 2$. With all hidden constraints the regularized DAE via over-determined formulation is given by equations (4a)-(4g). This regularized DAE consists of 7 equations for 5 unknowns $x = [p \ q \ v \ w \ \lambda]^T$ and, due to its over-determinedness, is not solvable within the MODELICA-framework, e.g., OpenModelica, Dymola, MapleSim. \triangleleft

3 The Software Package QUALIDAES

In the following, we consider over-determined quasi-linear DAEs of the form

$$M(x,t)\dot{x} = f(x,t), \quad (5a)$$

$$0 = g(x,t) \quad (5b)$$

on the domain $\mathbb{I} = [t_0, t_f]$ with initial values $x(t_0) = x_0 \in \mathbb{R}^n$, where $M \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^{m \times n})$ is called the *leading matrix* of the quasi-linear DAE and $f \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^{m \times n})$ as well as $g \in \mathcal{C}(\mathbb{R}^n \times \mathbb{I}, \mathbb{R}^{m_c})$ form its *right-hand side*.

Such over-determined formulations of the form (5) have currently the disadvantage that within the common MODELICA-frameworks it is impossible to model and integrate over-determined systems. Therefore, a direct numerical integrator for possibly over-determined formulations in form (5) has been implemented in the software package QUALIDAES which requires and exploits the quasi-linear structure of the model equations.

The software package QUALIDAES is suited for general over-determined quasi-linear DAEs of the form (5) with the assumption that the constraints (5b) are neither contradictory nor redundant, i.e.,

$$\text{rank} \left(\frac{\partial g}{\partial x}(x,t) \right) = m_c \quad (6)$$

for all consistent $(x,t) \in \mathbb{R}^n \times \mathbb{I}$. For a successful integration with QUALIDAES DAEs (5) with no hidden constraints are preferable. But often an integration of DAEs

(5) containing hidden constraints of level 1 at most is successful. In case of no hidden constraints it holds

$$\text{rank} \left(\begin{bmatrix} M(x,t) \\ \frac{\partial g}{\partial x}(x,t) \end{bmatrix} \right) = n \quad (7)$$

for all consistent $(x,t) \in \mathbb{R}^n \times \mathbb{I}$. Such over-determined formulations with no hidden constraints are obtained e.g. by application of the regularization approach described in the previous section.

The software package QUALIDAES is implemented in FORTRAN.

Certain features of QUALIDAES are to be emphasized which distinguish QUALIDAES from other solvers. Important is the fact that QUALIDAES respects all provided constraints. In particular, if no hidden constraints exist in (5), i.e., (7) holds, drift or instabilities are avoided during the numerical integration.

Interface for the model equations: The information of the model equations needed for the integration algorithm has to be provided in residual form, as following. The user or the calling subroutine has to provide the residual of the right hand side f for the differential part, the residual of the right hand side g for the constraint part, as well as the leading matrix M . All evaluated at a point (x,t) . Furthermore, there exists a rough graphical user interface in MATLAB (Higham and Higham (2005)) suited for model equations provided in MODELICA, see Altmeyer and Steinbrecher (2013).

Integration method: In QUALIDAES the 3-stage implicit Runge-Kutta Method Radau IIa of fixed order 5, see Hairer and Wanner (1996), as discretization of the overdetermined formulation is implemented.

As mentioned above, the code QUALIDAES offers the possibility to combine the discretization method with the regularization technique presented in the previous section. Therefore, the algorithm may use the over-determined regularization in form (5) as basis for the discretization. For more details on the discretization we refer to Steinbrecher (2006).

The discretization of the over-determined system (5) using the 3-stage Radau IIa method leads to an over-determined nonlinear stage equation of the form

$$0 = \begin{bmatrix} D(\xi_k) \\ C(\xi_k) \end{bmatrix} \quad \text{with} \quad \xi_k = \begin{bmatrix} X_{k1} \\ X_{k2} \\ X_{k3} \end{bmatrix} \quad (8)$$

for the determination of the three stages $X_{ki} \in \mathbb{R}^n$, $i = 1, 2, 3$ on the current integration interval $[t_k, t_{k+1}]$ with $t_{k+1} = t_k + \delta_k$. Here δ_k denotes the current step size. The stages $X_{ki} \in \mathbb{R}^n$, $i = 1, 2, 3$ approximate the solution at the points $t_{ki} = t_k + c_i \delta_k$. In (8) D represents the discretization of the differential part and C represents the discretization of the constraints to determine the next iterate x_{k+1} from ξ_k . Unfortunately, the nonlinear system

(8) is no longer solvable because of discretization and rounding errors. Therefore, it is only possible to find an approximation $\tilde{\xi}_k$ which minimizes the residual r of size $3(m_D + m_C)$ of the discretized over-determined DAE in a certain sense with

$$r = \begin{bmatrix} r_D \\ r_C \end{bmatrix} = \begin{bmatrix} D(\xi_k) \\ C(\xi_k) \end{bmatrix}.$$

In general, such an approximation yields a residual $r_C \neq 0$, which in turn leads to unfulfilled constraints, i.e., $C(\xi_k) \neq 0$ and, therefore, also $g(x_k, t_k) \neq 0$, 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 (8) is treated separately such that $\tilde{\xi}_k$ satisfies the lower part, i.e., the constraints, exactly or within a prescribed precision, while $\tilde{\xi}_k$ yields a minimal residual in the upper part, i.e., in the differential part.

For solving (8) as described above, an adaptation of a simplified Newton method is implemented in QUALIDAES. For more details on Newton methods we refer to Deuffhard (2004). In particular, a constant Newton iteration matrix is used for a certain number of Newton iteration steps inside the current integration step $[t_k, t_{k+1}]$. The usage of the simplified Newton method saves evaluation of Jacobians and decomposition of the Newton iteration matrix in every except the first Newton iteration step. Therefore, during the Newton iteration a linear system of the form

$$\begin{bmatrix} J_D \\ J_C \end{bmatrix} \Delta^j = \begin{bmatrix} d(\xi_k^j) \\ c(\xi_k^j) \end{bmatrix} \quad (9)$$

has to be solved in each Newton iteration step $j = 0, 1, \dots$ to obtain the next iteration $\tilde{\xi}_k^{j+1} = \tilde{\xi}_k^j + \Delta^j$ in the Newton iteration. The upper part in (9) represents the differential part while the lower part represents the constraint part. The solving of the linear algebraic system (9) has to be done in an efficient but stable way. For that the code QUALIDAES decomposes the differential part and the algebraic part via different decomposition methods. The LU decomposition with full pivoting is used for the constraint part and the LU decomposition with partial pivoting is used for the differential part. While the first full pivoting detects the set of locally constrained state variables the second decomposition is faster and detects a minimal set of differential equations for the locally dynamic state variables. For a $J \in \mathbb{N}$ we accept $\tilde{\xi}_k^J$ as the numerical solution of (8) if a certain stopping criteria of the Newton iteration is satisfied. Furthermore, from this $\tilde{\xi}_k^J$ we determine the next iterate x_{k+1} as approximation of the solution at t_{k+1} .

In particular, this strategy leads to a (numerically) precise fulfillment of the constraints while solving the differential part in an "approximate sense". For more details see also Scholz and Steinbrecher (2014).

Further features of QUALIDAES: The numerical integration implemented in QUALIDAES uses a variable step size strategy. For that an adaptation of the error estimation and the step size control implemented in the code RADAU5 is used in QUALIDAES.

Furthermore, QUALIDAES offers the possibility to check and (if necessary) to correct initial values. For that the user or the calling subroutine has to provide further initial conditions in addition to the provided constraints (5b).

If the model equations have solution invariants, e.g., energy conservation or mass conservation, then it is often desirable to preserve these solution invariants explicitly because in general the numerical solution of the model equations does not satisfy the solution invariants. QUALIDAES is able to preserve solution invariants if they are provided by the user as additional equations in the constraints (5b).

Furthermore, QUALIDAES offers the possibility to determine a continuous output. This is helpful for example for an event detection or a visualization in the post processing.

If QUALIDAES is used in the MODELICA-framework then this requires a representation of the MODELICA model equation in quasi-linear form. For the most real applications, the model equations arise naturally in quasi-linear form. But unfortunately, in the MODELICA-framework this quasi-linear structure is not obviously reflected. Therefore, it is necessary to develop strategies to represent MODELICA model equations in quasi-linear form or to reformulate, e.g., by extension into this structure, as illustrated in the next section.

4 Quasi-Linear Model Equations

As already mentioned, MODELICA does not support quasi-linear equations directly, but allows the user to write arbitrary expressions to describe the dynamic behavior of the model. It is the responsibility of the underlying interpreter to transform the expressions into an equivalent suitable form (or, arguably, report an error if no such transformation can be found). Therefore, to use MODELICA as the model language for QUALIDAES, we have to provide said transformation.

In the following section we will resort to the following style of notation:

A language will be defined in a simple BNF-form: Nonterminals are expressed with the same small letters as meta-variables of the corresponding syntactic sort (e.g. we will use e to denote both the set of terms and a variable from that set). Productions are defined by ::= and

alternatives are distinguished via $|$. The context will allow for easy distinction between both uses. Multiple variables of the same syntactic sort are introduced before they are used.

We will introduce the signature of functions in a blackboard style using \times for Cartesian products and \rightarrow to distinguish domain and co-domain. Partial functions will be introduced using the same style but with a \hookrightarrow . Additionally, we consider partial functions as sets of tuples that can be augmented using the \mapsto operator (i.e. $p \cup \{1 \mapsto 2\}$ is the partial function p augmented by mapping 1 to 2. The domain of a partial function p is written $\text{dom}(p)$).

Functions over elements of a language will be defined using a freely adapted denotational style: $\llbracket e \rrbracket_X$ denotes the function X applied to e . All these (recursive) functions are defined using pattern matching on their arguments: $\llbracket e_1 + e_2 \rrbracket_X$ means the application of X to terms formed by the addition of two (possible distinct) terms. Meta-variables are bound in the patterns or corresponding where-clauses.

4.1 Input Language

As input we consider a small excerpt from the MODELICA abstract syntax of terms:

$$\begin{array}{l|l|l}
 e ::= & e + e & | \quad e \times e \\
 & u_i & | \quad \text{DER}(u_i) \\
 & \tau & | \quad c
 \end{array}$$

Terms e (alternative variables are d, c) consist of addition, multiplication, numbered unknowns (u_i) a MODELICA-style derivative-operator $\text{DER}()$, the simulation time τ and constants $c \subseteq \mathbb{R}$.

4.2 Quasi-Linear Language

A quasi-linear equation ql (or $\hat{q}l, \tilde{q}l$) forms one row of the aforementioned $E(x, t)$. Without loss of generality, we assume that all ql are of the form $e_i(x, t)\dot{x} + k_i(x, t) = 0$. We also assume that all our system consists of n unknowns $u_1 \dots u_n = x$. Then, a quasi-linear equation can be represented as a tuple of a constant term e and a partial function γ (alternatively β, α), mapping derivatives to their respective coefficients:

$$\begin{array}{l}
 ql \subseteq \gamma \times e \\
 \text{where } \gamma : u \hookrightarrow e
 \end{array}$$

An ordered set $QL = \{ql_1 \dots ql_n\}$ ($\tilde{Q}L$ when an alternative is needed) of n quasi-linear equations forms a *system*. Such a system is equivalent to the leading matrix E augmented with its right-hand-side k in the sense that each

quasi-linear equation defines a row of $E|k$:

$$\begin{array}{l}
 QL = \{ql_1 \dots ql_n\} \stackrel{\Delta}{=} E|k \\
 e_{ij}(\dot{x}, x, t) \stackrel{\Delta}{=} c_{ij} \cdot \dot{x}
 \end{array}$$

where

$$\begin{array}{l}
 ql_i = \langle \gamma_i, e_i \rangle \\
 c_{ij} = \begin{cases} \llbracket \gamma_i(u_j), x, t \rrbracket_e & \text{when } u_j \in \text{dom}(\gamma_i) \\ 0 & \text{otherwise} \end{cases} \\
 k_i(x, t) = \llbracket e_i, x, t \rrbracket_e
 \end{array}$$

In this definition, the helper function $\llbracket \dots \rrbracket_e$ is a straightforward interpretation of terms:

$$\begin{array}{l}
 \llbracket \cdot \rrbracket_e : e \times \mathbb{R}^n \times \mathbb{R} \hookrightarrow \mathbb{R} \\
 \llbracket e_1 \times e_2, x, t \rrbracket_e \stackrel{\Delta}{=} \llbracket e_1, x, t \rrbracket_e \llbracket e_2, x, t \rrbracket_e \\
 \llbracket e_1 + e_2, x, t \rrbracket_e \stackrel{\Delta}{=} \llbracket e_1, x, t \rrbracket_e + \llbracket e_2, x, t \rrbracket_e \\
 \llbracket \tau, x, t \rrbracket_e \stackrel{\Delta}{=} t \\
 \llbracket u_i, x, t \rrbracket_e \stackrel{\Delta}{=} x_i \\
 \llbracket c, x, t \rrbracket_e \stackrel{\Delta}{=} c
 \end{array}$$

Note, that the interpretation function is undefined for derivative-terms. However, this does not cause any problems in our application, as any derivatives will be removed by our transformation (the matrix E does not contain any derivatives).

4.3 Transformation

With the above definitions, the remaining problem is how to transform a general MODELICA-style equation into a quasi-linear form. Naturally, there is a trivial transformation that replaces all derivative with simple identities of the form $\text{DER}(u_i) = u_j$. Since these identities are trivially quasi-linear, this does not violate the requirements for the output of the transformation. However, the resulting system would be unnecessary large and not leverage the structure of the system for efficient simulation. In fact, QUALIDAES would have to solve the whole nonlinear system as hidden constraints. While the result (if it can be computed) might be (numerically) exact, this is certainly not the best or even an acceptable strategy.

Instead, we are going to keep the amount of additional identities to a minimum. We capture the identities in a partial function ι (also: κ, λ):

$$\iota : u_i \hookrightarrow u_j$$

The transformation $\llbracket \dots \rrbracket_{qlt}$ itself is again defined in a denotational style using pattern-matching:

$$\llbracket \cdot \rrbracket_{qlt} : e \times \mathbb{N} \times \iota \rightarrow ql \times \mathbb{N} \times \iota$$

The simplest cases are the simulation time, constants and unknowns. In these cases, the result is the quasi-linear equation with an empty set of coefficients, while

the size of the system and the identities remain unchanged:

$$\begin{aligned} \llbracket \tau, n, \iota \rrbracket_{qlt} &\hat{=} \langle \langle \emptyset, \tau \rangle, n, \iota \rangle \\ \llbracket c, n, \iota \rrbracket_{qlt} &\hat{=} \langle \langle \emptyset, c \rangle, n, \iota \rangle \\ \llbracket u_i, n, \iota \rrbracket_{qlt} &\hat{=} \langle \langle \emptyset, u_i \rangle, n, \iota \rangle \end{aligned}$$

For the summation of quasi-linear equations, we need a way to sum their coefficients in a way that maintains the interpretation of the coefficients as products of derivatives with terms. Hence, the sum of two coefficients is either undefined for a given unknown (when both summands are undefined for the unknown), the result of looking up that unknown in one of the coefficients (when it is only defined in one of them, mimicking addition with zero) or the sum of both right-hand sides (when it is defined in both coefficients):

$$(\gamma \uplus \beta)(u_i) \hat{=} \begin{cases} \gamma(u_i) + \beta(u_i) & \text{when } u_i \in \text{dom}(\gamma) \cap \text{dom}(\beta) \\ \gamma(u_i) & \text{when } u_i \in \text{dom}(\gamma) \setminus \text{dom}(\beta) \\ \beta(u_i) & \text{when } u_i \in \text{dom}(\beta) \setminus \text{dom}(\gamma) \\ \text{undefined} & \text{otherwise} \end{cases}$$

With this definition, the transformation of the addition-term is the simple in-order transformation of both summands, followed by the construction of the quasi-linear sum:

$$\begin{aligned} \llbracket e_1 + e_2, n, \iota \rrbracket_{qlt} &\hat{=} \langle \gamma \uplus \beta, d + c, \rangle, l, \lambda \rangle \\ \text{where} & \\ \langle \langle \gamma, d \rangle, m, \kappa \rangle &= \llbracket e_1, n, \iota \rrbracket_{qlt} \\ \langle \langle \beta, c \rangle, l, \lambda \rangle &= \llbracket e_2, m, \kappa \rrbracket_{qlt} \end{aligned}$$

In order to transform a derivative, we have to check, whether said derivative is already identified with an (artificial) variable. In such a case, the derivative is replaced with the corresponding unknown and put into the right-hand-side term. If the derivative is not yet identified with another unknown, it yields a new coefficient:

$$\llbracket \text{DER}(u_i), n, \iota \rrbracket_{qlt} \hat{=} \begin{cases} \langle \langle \emptyset, u_{1(i)} \rangle, n, \iota \rangle & \text{when } i \in \text{dom}(\iota) \\ \langle \langle \{u_i \rightarrow 1\}, 0 \rangle, n, \iota \rangle & \text{when } i \notin \text{dom}(\iota) \end{cases}$$

The transformation of multiplication-terms requires another auxiliary function, $\llbracket \dots \rrbracket_{\times}$:

$$\begin{aligned} \llbracket e_1 \times e_2, n, \iota \rrbracket_{qlt} &\hat{=} \llbracket qlt, e_2, m, \kappa \rrbracket_{\times} \\ \text{where} & \\ \langle qlt, e_2, m, \kappa \rangle &= \llbracket e_1, n, \iota \rrbracket_{qlt} \end{aligned}$$

$\llbracket \dots \rrbracket_{\times}$ allows to multiply a quasi-linear equation directly with a term. Again, it is defined in a denotational style using pattern-matching:

$$\llbracket \dots \rrbracket_{\times} : ql \times e \times \mathbb{N} \times \iota \rightarrow ql \times \mathbb{N} \times \iota$$

Multiplication with non-derivative terms is again straight-forward (with \otimes being the multiplicative equivalent to \uplus defined above).

$$\begin{aligned} \llbracket \langle \gamma, e \rangle, \tau, n, \iota \rrbracket_{\times} &\hat{=} \langle \langle \gamma \otimes \tau, e \times \tau \rangle, n, \iota \rangle \\ \llbracket \langle \gamma, e \rangle, c, n, \iota \rrbracket_{\times} &\hat{=} \langle \langle \gamma \otimes c, e \times \tau \rangle, n, \iota \rangle \\ \llbracket \langle \gamma, e \rangle, u_i, n, \iota \rrbracket_{\times} &\hat{=} \langle \langle \gamma \otimes u_i, e \times \tau \rangle, n, \iota \rangle \end{aligned}$$

An addition-term can be multiplied with a quasi-linear equation by multiplying its summands and summing up the result:

$$\llbracket ql, e_1 + e_2, n, \iota \rrbracket_{\times} \hat{=} \langle \langle \gamma \uplus \beta, e + d \rangle, l, \lambda \rangle$$

where

$$\begin{aligned} \langle \langle \gamma, e \rangle, m, \kappa \rangle &= \llbracket ql, e_1, n, \iota \rrbracket_{\times} \\ \langle \langle \beta, e \rangle, l, \lambda \rangle &= \llbracket ql, e_2, m, \kappa \rrbracket_{\times} \end{aligned}$$

Multiplication with a derivative is uncomplicated, when there is no coefficient mapped to a derivative in the quasi-linear equation:

$$\llbracket \langle \emptyset, e \rangle, \text{DER}(u_i), n, \iota \rrbracket_{\times} \hat{=} \langle \langle \{u_i \rightarrow e\}, 0 \rangle, n, \iota \rangle$$

If the derivative is identified with another unknown, multiplication is defined recursively by multiplication with that unknown. In the general case, however, the multiplication with a derivative requires the addition of a new identification:

$$\begin{aligned} \llbracket ql, \text{DER}(u_i), n, \iota \rrbracket_{\times} &\hat{=} \\ \begin{cases} \llbracket ql, \iota(u_i), n, \iota \rrbracket_{\times} & \text{when } u_i \in \text{dom}(\iota) \\ \llbracket ql, u_m, m, \iota \cup \{u_i \rightarrow u_m\} \rrbracket_{\times} & \text{otherwise} \end{cases} \end{aligned}$$

where $m = n + 1$

The final case is the multiplication of multiplication-terms. In that case, we can simply resort to the distributive property of multiplication:

$$\llbracket ql, e_1 \times e_2, n, \iota \rrbracket_{\times} \hat{=} \langle \tilde{ql}, l, \lambda \rangle$$

where

$$\begin{aligned} \langle \hat{ql}, m, \kappa \rangle &= \llbracket ql, e_1, n, \iota \rrbracket_{\times} \\ \langle \tilde{ql}, l, \lambda \rangle &= \llbracket \hat{ql}, e_2, m, \kappa \rrbracket_{\times} \end{aligned}$$

This transformation obviously deals just with a tiny fraction of the syntactically valid MODELICA equations and it is also quite obvious (at least to the experienced developer), that a lot of work needs to be put into a full coverage. However, adding more operators or syntactic variants does not add anything more insight into the discussed principles. On the contrary, if we would add an operation like MODELICA's power-operator \wedge , we would have to expand our transformation with a corresponding $\llbracket \dots \rrbracket_{\wedge}$ routine. It should be quite clear that just a few such additions would make the transformation process unreadable. Hence, we conjecture (but do not prove for practical reasons) that *all* MODELICA equations can, in principle, be transformed into an equivalent quasi-linear form.

4.4 Derivatives and Hidden Constraints

The regularization (or index-reduction) of a DAE requires the (arbitrary-order) differentiation of equations with respect to the independent variable. For a quasi-linear representation however, the coefficients may only be multiplied with first-order derivatives of the system's unknowns. To maintain that invariant, it is necessary to break down an arbitrary-order differentiation into several steps of first-order differentiation and transformation into quasi-linear form.

The (first-order) derivative of a quasi-linear equation is computed by $\llbracket \cdot \rrbracket_{\nabla ql}$. It takes a quasi-linear equation, system size and identities and yields a term (which may contain applications of the $\text{DER}()$ -operator) and a new system size m and identities κ :

$$\begin{aligned} \llbracket \cdot \rrbracket_{\nabla ql} & : ql \times \mathbb{N} \times \iota \rightarrow e \times \mathbb{N} \times \iota \\ \llbracket \langle \gamma, e \rangle, n, \iota \rrbracket_{\nabla ql} & \triangleq \langle d + \llbracket e \rrbracket_{\nabla e}, m, \kappa \rangle \\ \text{where} & \end{aligned}$$

$$\langle d, m, \kappa \rangle = \llbracket \gamma, n, \iota \rrbracket_{\nabla \gamma}$$

The total derivative of the set of coefficients $\llbracket \cdot \rrbracket_{\nabla \gamma}$ is the sum of the total derivative of every coefficient. A coefficient can be differentiated by interpreting it as the product of the derivative with a term. To avoid generating a higher-order derivative, the coefficient's derivative has to be identified with a new or existing variable:

$$\begin{aligned} \llbracket \cdot \rrbracket_{\nabla \gamma} & : \gamma \times \mathbb{N} \times \iota \rightarrow e \times \mathbb{N} \times \iota \\ \llbracket \emptyset, n, \iota \rrbracket_{\nabla \gamma} & \triangleq \langle 0, n, \iota \rangle \\ \llbracket \{u_i \mapsto e\} \cup \gamma, n, \iota \rrbracket_{\nabla \gamma} & \triangleq \langle c, m+1, \kappa \cup \{u_i \mapsto u_{m+1}\} \rangle \\ \text{where} & \end{aligned}$$

$$\begin{aligned} c & = \llbracket e \rrbracket_{\nabla e} \times u_{m+1} + e \times \text{DER}(u_{m+1}) + d \\ \langle d, m, \kappa \rangle & = \llbracket \gamma, n, \iota \rrbracket_{\nabla \gamma} \end{aligned}$$

Calculating the total derivative of a (derivative-free) term is a straightforward implementation of calculus:

$$\begin{aligned} \llbracket \cdot \rrbracket_{\nabla e} & : e \mapsto e \\ \llbracket e_1 + e_2 \rrbracket_{\nabla e} & \triangleq \llbracket e_1 \rrbracket_{\nabla e} + \llbracket e_2 \rrbracket_{\nabla e} \\ \llbracket e_1 \times e_2 \rrbracket_{\nabla e} & \triangleq \llbracket e_1 \rrbracket_{\nabla e} \times e_2 + e_1 \times \llbracket e_2 \rrbracket_{\nabla e} \\ \llbracket c \rrbracket_{\nabla e} & \triangleq 0 \\ \llbracket \tau \rrbracket_{\nabla e} & \triangleq 1 \\ \llbracket u_i \rrbracket_{\nabla e} & \triangleq \text{DER}(u_i) \end{aligned}$$

Again, it comes in handy that our term language is so small. However, the above function can be generalized for more complicated input languages using techniques like automatic differentiation (Höger (2013)). Hence we conjecture that differentiation is possible for *all* quasi-linear equations derived from *all* MODELICA equations.

In order to calculate the constraints of a regularized DAE we consider the output of the regularization as a

function $\mathbf{c} : ql \rightarrow \mathbb{N}$ from quasi-linear equations to the amount of desired differentiations. Given such a function, a system of quasi-linear equations QL can be expanded by regularization $\llbracket \cdot \rrbracket_{\text{reg}}$:

$$\begin{aligned} \llbracket \cdot \rrbracket_{\text{reg}} & : (ql \rightarrow \mathbb{N}) \times QL \times \mathbb{N} \times \iota \rightarrow QL \times \mathbb{N} \times \iota \\ \llbracket \mathbf{c}, \emptyset, n, \iota \rrbracket_{\text{reg}} & \triangleq \langle \emptyset, n, \iota \rangle \\ \llbracket \mathbf{c}, \{ql\} \cup QL, n, \iota \rrbracket_{\text{reg}} & \triangleq \langle \{ql_0 \dots ql_k\} \cup \tilde{QL}, m, \kappa \rangle \\ \text{where} & \end{aligned}$$

$$\begin{aligned} k & = \mathbf{c}(ql) \\ \langle ql_0, n_0, \iota_0 \rangle & = \langle ql, n, \iota \rangle \\ \langle ql_{i+1}, n_{i+1}, \iota_{i+1} \rangle & = \llbracket \llbracket ql_i, n_i, \iota_i \rrbracket_{\nabla ql} \rrbracket_{qlt} \\ \langle \tilde{QL}, m, \kappa \rangle & = \llbracket \mathbf{c}, QL, n_k, \iota_k \rrbracket_{\text{reg}} \end{aligned}$$

After this process, the resulting augmented matrix $\tilde{E}|k$ (including rows from identities) is probably non-squared. To reconcile this property and gather all hidden constraints, we attempt to eliminate superfluous rows from the matrix e.g. by symbolic Gaussian elimination. This reconciliation depends on the symbolic equivalence of equation terms. While this is possible (e.g. by using a suitable computer algebra system) for our small term-language, equivalence is of course undecidable for Turing-complete terms (as they are used in MODELICA). Hence, the process is not practically applicable to *every* model. We conjecture however, that such a limitation exists for every symbolic processing of models.

If this process succeeds, all the removed rows have no coefficients and are thus hidden constraints.

4.5 Example

To support our claim that $\llbracket \cdot \rrbracket_{qlt}$ is not a trivial and hence useless transformation, we resort to example 2.1. After setting $m = L = 1$ for simplification, it can be expressed in our simple term language (extended with subtraction for brevity) as:

$$\begin{aligned} & \text{DER}(u_1) - u_3 \\ & \text{DER}(u_2) - u_4 \\ & u_1 \times u_1 + u_2 \times u_2 - 1 \\ & \text{DER}(u_3) + 2 \times u_1 \times u_5 \\ & \text{DER}(u_4) + 2 \times u_2 \times u_5 + g \end{aligned}$$

The first and second equation are obvious identities in the sense of ι . Hence, our transformation can be jump-started (if we omit this jump-start, the identities would be copied later on) using these identities and yields 3 quasi-linear equations and said identities:

$$\begin{aligned} ql_1 & = \langle \emptyset, u_1 \times u_1 + u_2 \times u_2 - 1 \rangle \\ ql_2 & = \langle \{u_3 \mapsto 1\}, 2 \times u_1 \times u_5 \rangle \\ ql_3 & = \langle \{u_4 \mapsto 1\}, 2 \times u_2 \times u_5 + g \rangle \\ \iota & = \{u_1 \mapsto u_3, u_2 \mapsto u_4\} \end{aligned}$$

The output of a regularization step will ask us to add the first and second derivative of equation ql_1 to the system (it will also ask us to add the first derivative of the identities, but this can be ignored since identities and their derivatives are inlined implicitly). Doing so yields:

$$ql_4 = \langle \emptyset, u_1 \times u_3 + u_2 \times u_4 \rangle$$

$$ql_5 = \langle \{u_3 \mapsto u_1, u_4 \mapsto u_2\}, u_3 \times u_3 + u_4 \times u_4 \rangle$$

No additional identities are required for this simple example. Equations ql_1 and ql_4 are arguably constraints (although ql_1 is not hidden). The resulting augmented coefficient matrix can be seen below:

$$\left[\begin{array}{ccc|ccc} u_3 & & & & & 0 \\ & u_4 & & & & 0 \\ & & 1 & & & 2 \times u_1 \times u_5 \\ & & & 1 & & 2 \times u_2 \times u_5 + g \\ & & u_1 & u_2 & & u_3 \times u_3 + u_4 \times u_4 \end{array} \right]$$

The last row can be eliminated by subtracting the third and fourth rows (multiplied with the corresponding coefficient), which yields the third hidden constraint:

$$u_3 \times u_3 + u_4 \times u_4 - 2 \times u_1 \times u_1 \times u_5 - 2 \times u_2 \times u_2 \times u_5 + g$$

This is precisely the term we would expect from translating equation (4g). The resulting system E, k and h can be fed into `QUALIDAES` and (given a consistent initial point) integrated over time without any further state-selection.

5 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 over-determined system formulation that is obtained by adding all hidden constraints explicitly to the original model equation. The over-determined system formulation can then directly be integrated using the software package `QUALIDAES`. The great advantage of the direct discretization of the over-determined formulation is the fact that it is not necessary to determine a dynamic (state) selector outside of the solver `QUALIDAES` since this is achieved automatically within the separated treatment of (8) by its numerical solution, described above. 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 Pepper et al. (2011). Furthermore, the number of unknowns in the DAE is not increased. A further advantage of an over-determined 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.

Nevertheless, `QUALIDAES` requires a quasi-linear representation of the model equations. As we have shown, `MODELICA`-style equations can be transformed into quasi-linear form in a non-trivial way. This transformation preserves enough symbolic information about the equations to allow for the description of the hidden constraints. On the other hand, non-symbolic (i.e. algorithmic) parts can still be dealt with due to the introduction of identities with new variables.

5.1 Future Work

Although clearly necessary, the expansion of the input language of the quasi-linear transformation seems to be merely technically challenging. There are however some areas of future research that should be considered:

First, the search for a consistent initial point is a well-known challenging problem. It remains an open question, whether the quasi-linear transformation could provide any help in that area.

Furthermore, the transformations are currently implemented in a straightforward manner. Hence, the outcome might be non-optimal for practical applications (e.g. the size of the derived expressions might harm the simulation performance). It could be interesting to search for variants of the transformation that maintains practically useful properties (e.g. minimal tree size, minimal identities added).

Finally, we have already shown that the regularization of a structurally varying DAE can be implemented in an efficient, dynamic algorithm (see Höger (2014)). This is, obviously, of little value when the application of its results remains a non-dynamic monolithic algorithm. Hence any representation, but especially the quasi-linear form (since it is well-suited for structurally varying systems) should be enhanced with a *dynamic* regularization that preserves as much information from earlier modes as possible.

Acknowledgments

This work has been supported by the European Research Council through Advanced Grant *MODSIMCONMP* and by the German Research Foundation (Deutsche Forschungsgemeinschaft DFG) within the project "Automatische Modellierung und Simulation von technischen Systemen mit Unsicherheiten" *AMSUN*.

References

R. Altmeyer and A. Steinbrecher. Regularization and numerical simulation of dynamical systems modeled with Model-

- ica. Preprint 29-2013, Institut für Mathematik, TU Berlin, 2013.
- 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.
- P. Deuffhard. *Newton methods for nonlinear problems. Affine invariance and adaptive algorithms*, volume 35 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2004.
- C.W. Gear. Differential-algebraic equation index transformations. *SIAM Journal on Scientific and Statistic Computing*, 9:39–47, 1988.
- 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.
- E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin, Germany, 2nd edition, 1996.
- D.J. Higham and N.J. Higham. *MATLAB Guide*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2005. ISBN 0-89871-578-4.
- C. Höger. Operational semantics for a modular equation language. *AVICPS 2013*, page 5, 2013.
- C. Höger. Dynamic structural analysis for daes. In *Proceedings of the 2014 Summer Simulation Multiconference*, page 12. Society for Computer Simulation International, 2014.
- 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.
- P. Kunkel and V. Mehrmann. *Differential-Algebraic Equations. Analysis and Numerical Solution*. EMS Publishing House, Zürich, Switzerland, 2006.
- 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.
- C.C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistic Computing*, 9:213–231, 1988.
- 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, 2011. September 5, 2011.
- J. Pryce. A simple structural analysis method for DAEs. *BIT Numerical Mathematics*, 41:364–394, 2001.
- 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.
- L. Scholz and A. Steinbrecher. Efficient numerical integration of dynamical systems based on structural-algebraic regularization avoiding state selection. In K.-E. Arzen H. Tummescheit, editor, *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, number 96 in Linköping Electronic Conference Proceedings, pages 1171–1178. Modelica Association and Linköping University Electronic Press, 2014.
- A. Steinbrecher. *Numerical Solution of Quasi-Linear Differential-Algebraic Equations and Industrial Simulation of Multibody Systems*. PhD thesis, Technische Universität Berlin, 2006.

Fractional-Order Modelling in Modelica

Alexander Pollok¹ Dirk Zimmer¹ Francesco Casella²

¹Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
{alexander.pollok, dirk.zimmer}@dlr.de

²Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
francesco.casella@polimi.it

Abstract

Most dynamic systems with a basis in nature can be described using Differential-Algebraic Equations (DAE), and hence be modelled using the modelling language Modelica. However, the concept of DAEs can still be generalised, when differential operators of non-integer order are considered. These so called fractional order systems have counterparts in naturally occurring systems, for instance in electrochemistry and viscoelasticity. This paper presents an implementation of approximate fractional-order differential operators in Modelica, increasing the scope of systems that can be described in a meaningful way. Properties of fractional-order systems are discussed and some approximation methods are presented. An implementation in Modelica is proposed for the first time. Several testing procedures and their results are displayed. The work is then illustrated by the application of the model to several physically motivated examples. A possible usability-enhancement using the concept of "Calling Blocks as functions" is suggested.

Keywords: Fractional Order Systems, fractional calculus, Integer-Order Approximations

1 Introduction

In Modelica, models are represented as Differential-Algebraic Equations, i.e., equations of the form $F(\dot{x}(t), x(t), t) = 0$. This formulation is adequate for most physical systems that can be described (or at least approximated) with a finite number of states. There are, however, some systems, where a more general but ultimately similar framework is needed: If fractional derivatives occur, the traditional DAE formulation is inadequate.

Fractional calculus, a misnomer¹, is a branch of mathematics that deals with non-integer powers of differentiation operators. The introduction to this concept is much simpler in the Laplace-domain. Normally, the Laplace

¹this generalisation of differentiation operators is not restricted to fractions

variable is restricted to integer values. In fractional calculus, this restriction is lifted. Let us imagine the bode diagrams of the derivative operator ($s = s^1$), the unity operator ($1 = s^0$) and the half-derivator ($s^{0.5}$). The amplitude plot of the half-derivator has a slope of 10dB per decade, while the phase angle is constant at 45 degrees. This is illustrated in Figure 1. A detailed discussion of fractional calculus is given by Sabatier et al. (2007).

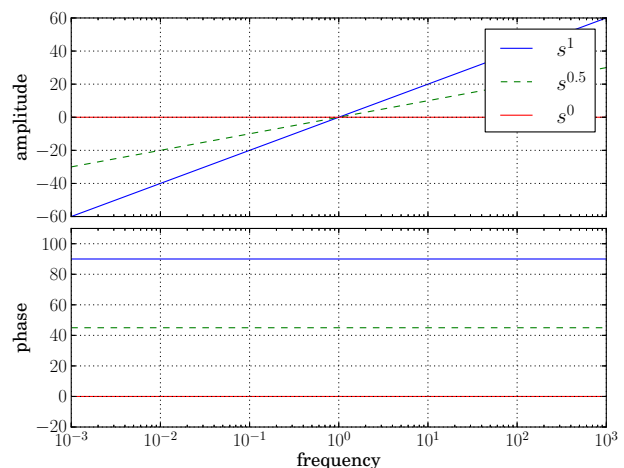


Figure 1. Illustration of fractional derivatives

Going back to time-domain, fractional differential operators can be defined in various ways, with the Caputo definition (Caputo, 1967) being applied in this paper. Contrary to the often preferred Riemann-Liouville definition, the Caputo definition allows for a physically meaningful initialisation of the operator. The Caputo Fractional Derivative of order α is defined as

$$\mathcal{L}^{-1}(s^\alpha) = D^\alpha f(t) := \frac{1}{\Gamma(m-\alpha)} \cdot \int_0^t \frac{f^m \tau}{(t-\tau)^{\alpha+1-m}} d\tau$$

with $f: \mathbb{R} \rightarrow \mathbb{R}$ being a continuous, differentiable function, the gamma function Γ , $\alpha \in \mathbb{R}, 0 < \alpha$ and $m \in \mathbb{Z}^+, m = \text{ceil}[\alpha]$.

Fractional-order systems occur naturally in various fields, like electrochemistry (Debnath, 2003), viscoelasticity (Koeller, 1984), heat diffusion (Povstenko, 2004) and biology (Magin, 2004). For example, exact solutions

for local temperature and heat flux at the boundary of a semi-infinite body, using fractional calculus, are given by Kulish and Lage (2000) (for a Modelica Standard Library (MSL)-friendly implementation of this findings see Subsection 3.1). Another application is shaping noise frequency content according to given spectra (Klößner et al., 2015). The usefulness is not limited to the modelling of PDE's though, fractional order modelling can be used to describe the dynamics of scale-free networks, for instance (Goodwine and Leyden, 2015).

The goal of this paper is to show how fractional-order systems can be modelled using Modelica. The derivation, implementation and testing of suitable approximations in Modelica is illustrated in Section 2. Applications of this implementations are shown in Section 3. At last, the contents of the paper are discussed and possible ramifications to the Modelica languages are addressed in Section 4.

2 Fractional Order Modelling

2.1 Implementation

In Modelica, only the `der()`-operator has to be generalized to model arbitrary fractional order systems. Unfortunately, defining a new operator `fracder(state, order)` is not possible based on the Modelica Language Specification 3.3. Therefore, the proposed implementation uses a Single-Input Single-Output block instead.

Fractional order systems have infinite dimensional transfer functions, and therefore have infinite memory (Vinagre et al., 2000). It is, however, possible to find reasonable approximations if the frequency range of interest is bounded (for a detailed error analysis refer to Pan and Das (2012)). Accordingly, as long as the modeller is not interested in extremely stiff systems, these approximations are adequate.

For implementations based on equation-based modelling languages, integer-order continuous approximations are the most useful. For examples, see Carlson and Halijak (1964), Xue et al. (2006) or Oustaloup et al. (2000). Some other methods are described in Vinagre et al. (2000), but not mentioned here, as their implementation in Modelica proved difficult due to a lack of user-level symbolic manipulation capabilities.

For Carlson's method, Oustaloup's method and Xue's method, we found general symbolic expressions for the approximating transfer-functions, and implemented them in Modelica. Preliminary analysis showed that Oustaloup's method was superior regarding flexibility and accuracy. For this reason, in the following only Oustaloup's method and its implementation details are presented.

The integer-order approximation of a fractional operator by Oustaloup's method is given in the Laplace do-

main by

$$s^\lambda \approx G(s) = \omega_h^\lambda \cdot \prod_{k=-N}^N \frac{s + \omega'_k}{s + \omega_k} \quad (1)$$

$$\omega_k = \omega_b \left(\frac{\omega_h}{\omega_b} \right)^{\frac{k+N+0.5(1+\gamma)}{2N+1}}, \omega'_k = \omega_b \left(\frac{\omega_h}{\omega_b} \right)^{\frac{k+N+0.5(1-\gamma)}{2N+1}} \quad (2)$$

with the fitting range (ω_b, ω_h) , the fraction of differentiation λ and the order of approximation N .

An important thing to notice is that λ is not bounded, so it is possible to simulate the second integral using $\lambda = -2$, for example. However, more accurate results can be obtained if $abs(\lambda)$ is kept low and surplus differential operations are simulated directly using the standard Modelica-notation.

We recreated the construction rule for the Oustaloup-Approximator in Modelica using linked first-order elements. The corresponding code can be seen in Listings 1, 2 and 3.

2.2 Testing

To test the validity and accuracy of the derived models, we applied three different testing scenarios: bode diagram, step response, and harmonic displacement. These tests are described in the following.

2.2.1 Bode Diagram

A Bode diagram of the unity operator (1 in the Laplace domain) is a straight line with amplitude 1 and phase angle zero degrees over the complete frequency range. The differentiator (s in the Laplace domain) has an positive slope of 20dB per decade and +90 degrees phase angle. Other operators like s^2 or s^{-1} behave analogous. From this, we require the half-differentiator $s^{0.5}$ to feature an ascending amplitude of 10dB per decade and +45 degrees phase angle.

In Figure 2 and Figure 3, bode plots of the implemented model with approximation orders 2 and 4 and fitting range (0.001Hz,1000Hz) are presented.

It can be seen that slope of the amplitude shows a close fit to the required 10dB per decade. The phase angle shows pronounced ripple effects in the case of the 2nd order approximation, but no visible ripples in the case of the 4th order approximation.

The required amplitude values are matched in the complete fitting range. For the phase angle the acceptable range is somewhat smaller.

If the fitting interval is increased to cover a broader range of frequencies (not shown here), noticeable ripples in the phase plot appear even for the 4th order approximation.

Listing 1. Excerpt of Modelica code for a fractional derivative operator

```

block OustaloupOperator
  import Modelica.Blocks.Types.Init;

  parameter Integer order(min=1, max=4) = 4 "order of approximation (1,2,3,4)";
  parameter Real lambda = 0.5 "exponent of operator (-1=integrator, 1=derivative)";
  parameter Real w_lower(max=1) = 0.001 "lower fitting frequency [1/s]";
  parameter Real w_upper(min=1) = 1000 "higher fitting frequency [1/s]";

  parameter Modelica.Blocks.Types.Init initType=Init.InitialState
    "Type of initialization (1: no init, 2: steady state, 3: initial state,
    4: initial output)" annotation(Evaluate=true, Dialog(group= "Initialization"));
  parameter Real x_start[number]=zeros(number) "Initial or guess values of states"
    annotation (Dialog(group="Initialization"));
  parameter Real y_start=0 "Initial value of output"
    annotation(Dialog(enable=initType == Init.InitialOutput, group= "Initialization"));

  final parameter Real wb = w_lower*Modelica.Constants.pi;
  final parameter Real wh = w_upper*Modelica.Constants.pi;
  final parameter Integer number = 1 + order*2;
  final parameter Real K = wh^(lambda);
  final parameter Real wk[number] =
    {FractionalOrder.Approximations.Internal.wk(i,wb,wh,order,lambda)
    for i in -order:order};
  final parameter Real wks[number] =
    {FractionalOrder.Approximations.Internal.wks(i,wb,wh,order,lambda)
    for i in -order:order};

  Real y_internal[number];
  Real x_internal[number];

  Modelica.Blocks.Interfaces.RealInput u
    annotation (Placement(transformation(extent={{-120,-10},{-100,10}})));
  Modelica.Blocks.Interfaces.RealOutput y
    annotation (Placement(transformation(extent={{100,-10},{120,10}})));

equation
  der(x_internal[1]) = -wk[1]*x_internal[1] + (wks[1]-wk[1]) * u*K;
  y_internal[1] = x_internal[1] + u*K;

  for i in 2:number loop
    der(x_internal[i]) = -wk[i]*x_internal[i] + (wks[i]-wk[i]) * y_internal[i-1];
    y_internal[i] = x_internal[i] + y_internal[i-1];
  end for;

  y = y_internal[number];

initial equation
  if initType == Init.SteadyState then
    der(x_internal) = zeros(number);
  elseif initType == Init.InitialState then
    x_internal = x_start;
  elseif initType == Init.InitialOutput then
    y = y_start;
  end if;

end OustaloupOperator;

```

Listing 2. First internal function for the generation of coefficients

```

function wks
  extends Modelica.Icons.Function ;
  input Integer k;
  input Real wb;
  input Real wh;
  input Real N;
  input Real lambda;
  output Real wks;
algorithm
  wks:=wb*(wh/wb)^((k+N+(1-lambda)/2)/(2*N+1));
end wks;
  
```

Listing 3. Second internal function for the generation of coefficients

```

function wk
  extends Modelica.Icons.Function ;
  input Integer k;
  input Real wb;
  input Real wh;
  input Real N;
  input Real lambda;
  output Real wk;
algorithm
  wk:=wb*(wh/wb)^((k+N+(1+lambda)/2)/(2*N+1));
end wk;
  
```

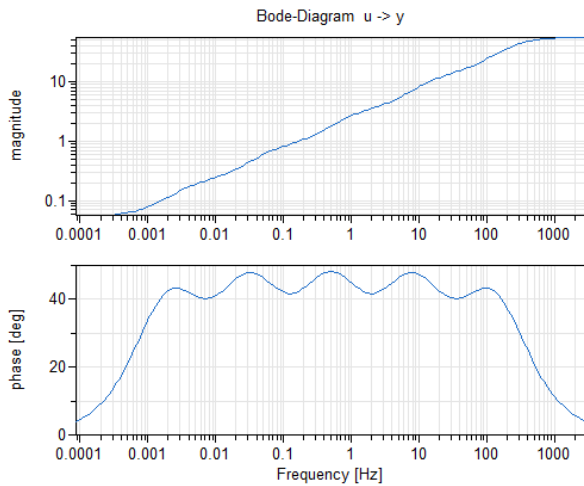


Figure 2. Bode diagram of 2nd order approximation of the half-derivative $s^{0.5}$ with fitting interval (0.001Hz,1000Hz)

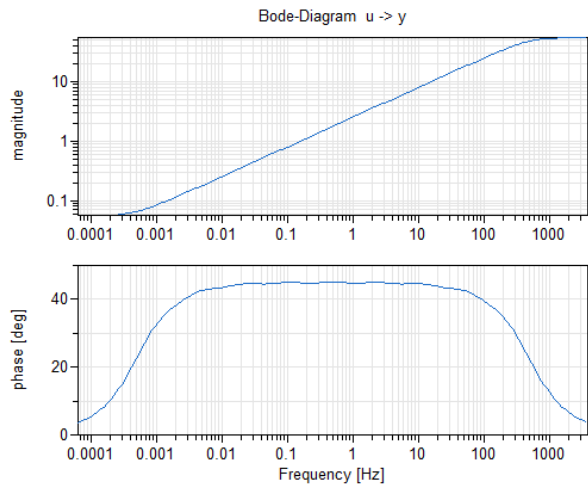


Figure 3. Bode diagram of the 4th order approximation of the half-derivative $s^{0.5}$ with fitting interval (0.001Hz,1000Hz)

2.2.2 Step Response

The step responses of the unity operator $y(t) = u(t)$ and the integrator $\dot{y}(t) = u(t)$ are known to be $y(t) = 1$ and $y(t) = t$ respectively, neglecting the initial conditions. Simultaneously, the unity operator and the integrator are identified in the Laplace Domain by s^0 and s^{-1} . The step responses of the fractional derivatives defined by s^λ with $-1 \leq \lambda \leq 0$ have to constitute the continuous transition

between those known step responses (Oldham, 1974).

The implemented model was instantiated 6 times and assigned λ -values in 0.2 intervals between -1 and 0. All models were subjected to a unit step (implemented by setting the input to 1 and the initial states of the models to 0). The results of this test can be seen in Figure 4.

It can be seen that the 6 step-responses form a smooth transition, and the outer ones correspond to the known

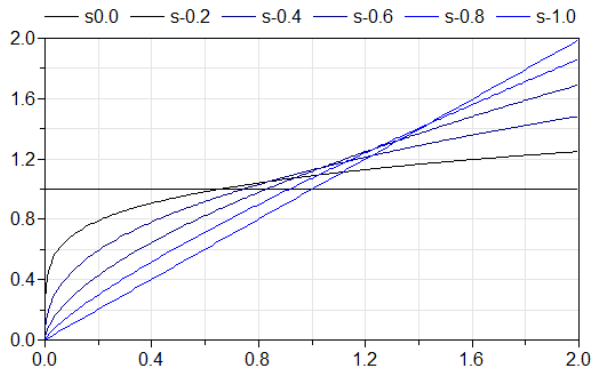


Figure 4. Step responses of $s^{0.0}$, $s^{-0.2}$, $s^{-0.4}$, $s^{-0.6}$, $s^{-0.8}$ and $s^{-1.0}$, approximated with the 3rd order Oustaloup's Method

step-responses mentioned earlier. All curves also match the curves given in Oldham (1974).

2.2.3 Harmonic Displacement

If a harmonic function is derived or integrated, the resulting function is a new harmonic function with a phase offset. For the second test, it is required that the result of fractionally integrated or derived harmonic functions behaves analogous.

The implemented model was instantiated 8 times and assigned λ -values in intervals of size 0.5 between -1 and 2.5. All models were subjected to a cosine input. Initialisation was done in such a way as to avoid unnecessary large initial transients: The models with derivative character would see the onset of the cosine input as a step and correspondingly respond with an impulse. For this reason, their initial states were set to a steady state solution. Models with integrative character were set to zero initial conditions, to set their integration constants to zero as well. The results of this test can be seen in Figure 5.

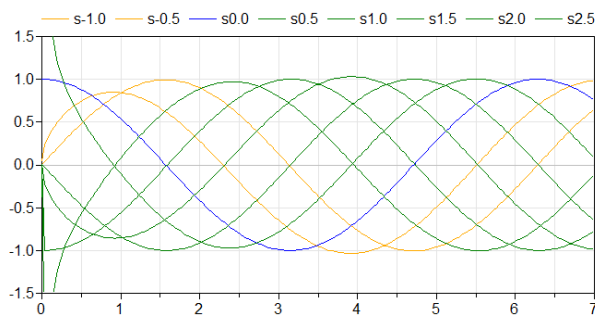


Figure 5. Cosine response of $s^{-1.0}$, $s^{-0.5}$, $s^{0.0}$, $s^{0.5}$ (zero state initialisation) and $s^{1.0}$, $s^{1.5}$, $s^{2.0}$, $s^{2.5}$ (steady state initialisation) approximated with the 3rd order Oustaloup's Method

It can be seen that all model outputs form harmonic functions. Also, the offsets between the functions are uniform. The initialisations of $s^{0.5}$ and $s^{2.5}$ are obviously not optimal, but after a few seconds those deviations vanish.

3 Examples

3.1 Heat Conduction

In Kulish and Lage (2000), relationships between temperature and heat flow rate at arbitrary locations in semi-infinite domains are developed. The temperature at a given time at the boundary is in this way given by

$$T(t) = \frac{\alpha^{1/2}}{2 \cdot A \cdot k} \cdot \frac{\delta^{-1/2} Q(t)}{\delta t^{-1/2}} + T_0 \quad (3)$$

with the thermal conductivity k , the thermal diffusivity α , the Area A , the heat flow rate Q , and the starting temperature T_0 .

As can be seen in Listing 4, the corresponding implementation in Modelica is straightforward and compact.

Listing 4. Modelica implementation of a semi-infinite thermal domain

```

Approximations.OustaloupOperator
  halfInt (order=3, lambda=-0.5);
equation
  halfInt.u = heatPort.Q_flow;
  heatPort.T =
    (alpha^(1/2)/(k*A*2)*halfInt.y) + T_0;

```

In Figure 6, the result of a simulation can be seen, where a semi-infinite block was subjected to a periodic rectangular heat flow rate at the boundary. The temperature at the boundary exhibits strong memory-effects, as would be expected from such a system.

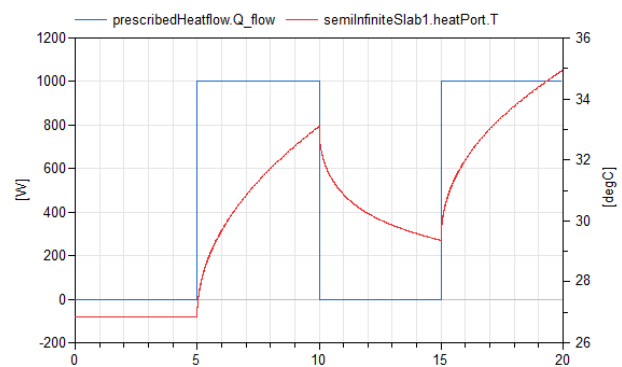


Figure 6. Temperature response of a semi-infinite domain subjected to periodic rectangular heat flow

3.2 Viscoelasticity

The dynamic behaviour of viscous fluids is commonly described with the Navier-Stokes equations. For the dynamic behavior of linear elastic materials, the Lamé-Navier equations are used. Both equations have some similarities. As an example, let us take a look at the respective relationships between stress/velocity and

stress/strain for incompressible fluids and linear-elastic solids:

$$\begin{aligned}\boldsymbol{\tau} &= \mu (\nabla \mathbf{v} + \nabla \mathbf{v}^T) \\ \boldsymbol{\sigma} &= \frac{1}{2} \mathbb{C} (\nabla \mathbf{u} + \nabla \mathbf{u}^T)\end{aligned}\quad (4)$$

with the stress tensors $\boldsymbol{\tau}$ and $\boldsymbol{\sigma}$, the viscosity μ , the tensor of elasticity \mathbb{C} , and the velocity and displacement tensors \mathbf{v} and \mathbf{u} . The structure of both equations essentially differs only by one differential operation, as the velocity is the derivative of the displacement w.r.t. time.

Likewise, a relationship between the stress and strain in one-dimensional viscoelastic materials was found in Stiassnie (1979).

$$\boldsymbol{\sigma} = k \cdot \frac{\delta^\alpha \boldsymbol{\varepsilon}}{\delta t^\alpha} \quad (0 \leq \alpha \leq 1) \quad (5)$$

with the stress $\boldsymbol{\tau}$, the material properties k and α , and the strain $\boldsymbol{\varepsilon}$. For α -values of 0 and 1, the behaviour of pure solids and fluids is obtained.

As with the other example, the implementation of this model in Modelica only takes two lines of code (not shown here).

The response of a viscoelastic block ($\alpha = 0.45$) under constant tension can be seen in Figure 7. The result is similar to the results presented by Stiassnie (1979), where the model is validated against real-world measurements.

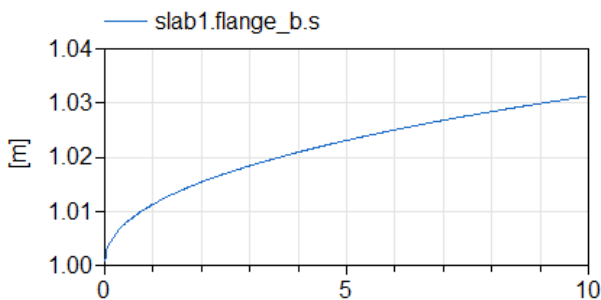


Figure 7. tension-response of a viscoelastic block with a length of 1m under constant tension

4 Discussion

The last sections showed that DAE systems containing time-derivatives of fractional order can be successfully implemented by the given means of the Modelica language. The provided solution offers an approximation that is good enough for at least a large set of technical applications. However, the given examples indicate that the typical application of fractional order derivatives is on the textual modelling level and that the applied formulation is, although practically feasible, still more clumsy

than actually necessary. The reason for this is the implementation in block-form. This leads to a declaration that has a dummy character since it is only being used to textually connect its input and output. Because any implementation of a fractional-order operator requires an internal state, an implementation as function is not possible. Yet, it would be very helpful for the modeller if he could call the block like it would be a function. In concrete terms, this means that an anonymous declaration of a block in the equation section is enabled whose inputs and outputs are connected within the declaration statement. To illustrate this mechanism, let us revisit the example of Listing 4:

Listing 5. Modelica implementation of a semi-infinite thermal domain using the "calling blocks as function"-approach

```
block halfInt =
  Approximations.OustaloupOperator
  (order=3, lambda=0.5);
equation
  heatPort.T = (alpha^(1/2)/k)
  * halfInt(u=heatPort.Q_flow).y/A + T_0;
```

Listing 5 presents a reformulation of Listing 3 based on the concept "Calling blocks as function". First, a local declaration of a half-integrator is created from the general Operator-block. Then the expression `halfInt(u=heatPort.Q_flow).y` is represented as an anonymous declaration of the halfInt block. Within the parantheses, the input is connected and the `.y` states that the expression as a whole represents the output signal of the block. The presented concept "Calling Blocks as Function" is not a new idea. Different syntactical variants are currently in discussion within the Modelica Association based on contributions by Martin Otter, Hans Olsson, Peter Fritzson, Michael Sasena, Martin Sjölund and others. An implementation variant in an experimental equation-based language can be found for instance in Sol (Zimmer, 2010). Should this feature become part of a future Modelica language version, modelling with fractional order time-derivatives will be almost as convenient as with standard time derivatives.

5 Conclusion

By design, the Modelica language is limited to the use of integer-order differential operators. This excludes the modelling of certain physical systems. We present an implementation of Oustaloup's approximation method in Modelica. The resulting model approximates fractional-order differential operators. Parameters for approximation order and frequency fitting range can be used to tailor the model to a specific application. In this way, the mentioned limitation of the Modelica language can be conveniently bypassed, thus increasing the scope of physical systems that can be described in a meaningful manner.

Reproducible research

The results of this paper can be reproduced using the code which is made available on:
github.com/DLR-SR/FractionalOrder

References

- Michele Caputo. Linear models of dissipation whose q is almost frequency independent part 2. *Geophysical Journal International*, 13(5):529–539, 1967.
- G Carlson and C Halijak. Approximation of fractional capacitors $(1/s)^{1/n}$ by a regular newton process. *Circuit Theory, IEEE Transactions on*, 11(2):210–213, 1964.
- Lokenath Debnath. Recent applications of fractional calculus to science and engineering. *International Journal of Mathematics and Mathematical Sciences*, 2003(54):3413–3442, 2003.
- Bill Goodwine and Kevin Leyden. Recent results in fractional-order modeling in multi-agent systems and linear friction welding. *IFAC-PapersOnLine*, 48(1):380–381, 2015.
- Andreas Klöckner, Andreas Knoblach, and Andreas Heckmann. How to shape noise spectra for continuous system simulation. In *Proceedings of the 11th International Modelica Conference*, 2015.
- RC Koeller. Applications of fractional calculus to the theory of viscoelasticity. *Journal of Applied Mechanics*, 51(2):299–307, 1984.
- VV Kulish and JL Lage. Fractional-diffusion solutions for transient local temperature and heat flux. *Transactions-American Society of Mechanical Engineers Journal of Heat Transfer*, 122(2):372–375, 2000.
- Richard L Magin. Fractional calculus in bioengineering. *Critical Reviews in Biomedical Engineering*, 32(1), 2004.
- Keith B Oldham. *The fractional calculus*. Elsevier, 1974.
- Alain Oustaloup, Francois Levron, Benoit Mathieu, and Florence M Nanot. Frequency-band complex noninteger differentiator: characterization and synthesis. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 47(1):25–39, 2000.
- Indranil Pan and Saptarshi Das. *Intelligent fractional order systems and control: an introduction*. Springer Publishing Company, Incorporated, 2012.
- Yu Z Povstenko. Fractional heat conduction equation and associated thermal stress. *Journal of Thermal Stresses*, 28(1): 83–102, 2004.
- J Sabatier, Om P Agrawal, and JA Tenreiro Machado. *Advances in fractional calculus*, volume 4. Springer, 2007.
- Michael Stiassnie. On the application of fractional calculus for the formulation of viscoelastic models. *Applied Mathematical Modelling*, 3(4):300–302, 1979.
- BM Vinagre, I Podlubny, A Hernandez, and V Feliu. Some approximations of fractional order operators used in control theory and applications. *Fractional calculus and applied analysis*, 3(3):231–248, 2000.
- Dingyu Xue, Chunna Zhao, and Yang Quan Chen. A modified approximation method of fractional order system. In *Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on*, pages 1043–1048. IEEE, 2006.
- Dirk Zimmer. *Equation-based modeling of variable-structure systems*. PhD thesis, Swiss Federal Institute of Technology, Zürich, 2010.

Modelica Library for Feed Drive Systems

Denis Özdemir Tobias Motschke Werner Herfs Christian Brecher

Laboratory for Machine Tools and Production Engineering (WZL) of RWTH Aachen University, Germany,
{D.Oezdemir, T.Motschke, W.Herfs, C.Brecher}@wzl.rwth-aachen.de

Abstract

As a part of machine tools and production machines, the primary task of feed drives is to create the contour of a workpiece by moving it and/or the tool along one or more axes according to the control input. This paper presents a Modelica library for feed drive systems consisting of electrical, electro-mechanical and mechanical components. The aim of the library is to provide engineers means to design feed drive models that can be parametrized with available data from component suppliers. The models are augmented with metrics and requirements to facilitate simulation analysis.

Keywords: Feed drives, servo motors, machine tools

1 Introduction

The design of feed drives systems is a complex technical problem due to the numerous requirements, design variables and interactions. Therefore, many computer-aided methods have been developed and simulation techniques are employed since the 1980s, e.g. (Simon 1986). These approaches mostly rely on signal-oriented models that are well suited to describe control structures (Brecher 2002; Zirn 2008). However, using the signal-oriented approach for physical systems such as multi-mass oscillators yields complicated structures that are difficult to understand even for experts. An alternative is to couple control-simulation with multi-body simulation or finite element software (Altintas et al. 2011). In industrial practice, however, these simulation approaches have hardly been applied as a recent survey shows (Brecher et al. 2014). While many machine suppliers see great potential in virtual prototypes, qualified personal and the costs of software are main obstacles.

In contrast to these simulation approaches motor sizing software such as Sizer by Siemens or Motion Analyzer by Rockwell Automation are widely used. These tools allow a quick selection of an adequate motor for a specified application, but the mechanical part of the system is assumed to be given and dynamic properties of the feed drive system as a whole are not taken into account.

The Modelica library for feed drive systems aims to close the gap between the elaborated simulation tech-

niques on the one hand and the sizing software on the other hand. This means the library provides models that can be used with limited expert knowledge by leveraging the concept of component-orientation in Modelica. Such an approach implies models that can be parametrized from available supplier data. In addition to the behavioral equations, metrics and requirements are included in the components to highlight critical behavioral aspects.

The Modelica library for feed drive systems is part of a planned design environment for feed drive systems where an optimizer is used to find those parameters and components from a database that optimally fulfill the requirements. The concept of the design environment and its potential implementation in the design process of machine tools is presented in an additional paper (Özdemir et al. 2015). In a preceding paper parts of an earlier version of the library have been presented briefly (Herfs et al. 2015). This paper therefore focuses on the newly developed aspects.

The contents of this paper fall into three main parts. In Chapter 2 models of feed drives motors are described. Chapter 3 addresses mechanical components and Chapter 4 presents results from the simulation of the system as a whole. Finally, Chapter 5 summarizes the results and explains how these models are embedded into the feed drive design environment.

2 Models for Feed Drive Motors

Feed drive motors of modern machine tools are primarily permanent-magnet synchronous motors (PSM), which are therefore the focus of this paper (s. Chap. 2.1). The models for permanent-magnet synchronous motors with field weakening option (s. Chap. 2.2) as well as for linear motors (s. Chap. 2.3) can be easily developed based on the model of the PSM. Hereby, the object-oriented approach has the advantage that many components can be reused.

2.1 Model for Permanently Excited Synchronous Motors (PSM)

The Modelica Standard Library contains two models of the PSM in the *Electrical.Machines* and *Magnetic.FundamentalWave* sublibraries. The core of these PSM models is a model of the air gap, which describes

the electromechanical torque as the cross product of current and magnetic flux in the dq-coordinate system that is fixed to the rotor (Kral, Haumer 2005; Kral 2011). While the PSM model in the Machines library uses space vectors for current, voltage and flux linkage the *FundamentalWave* model follows an even more physically rigorous approach with complex vectors. In the Machine library heat losses are considered with regards to the resistance of the stator windings and the damper cage. In addition eddy current losses in the stator core and mechanical losses depending on the speed are taken into account.

However, this rigorous physical approach requires a variety of physical parameters, e.g. the inductances in d- and q-direction, the leakage inductance of the stator and parameters for eddy current losses. Because these parameters are generally not available during drive selection, the PSM models of the Modelica Standard Library have to be simplified. Moreover, the standard PSM models do not include the design requirements that have to be considered during feed drive design.

2.1.1 Behavior Model

Typically available supplier data regarding the physical behavior is displayed in Table 1. Based on this data a standard model for the PSM (Schröder 2009, p. 394) can be implemented:

$$U_d = -p \cdot \omega_M \cdot L_D \cdot I_q, \quad (2)$$

$$U_q = L_D \cdot \frac{dI_q}{dt} + R \cdot I_q + p \cdot \omega_M \cdot \psi_{PM}, \quad (3)$$

$$M_M = 3 \cdot p \cdot \psi_{PM} \cdot I_q \quad (4)$$

Here, U_d , U_q and I_q are the RMS values of the motor voltage and torque building current in the dq-coordinate system and ω_M denotes the angular velocity of the rotor, see Figure 1. The number of pole pairs p , the effective inductivity L_D and the winding resistance R can be taken from the supplier data. The magnetic flux of the permanent-magnet ψ_{PM} relates motor torque M_M and torque-building current I_q and

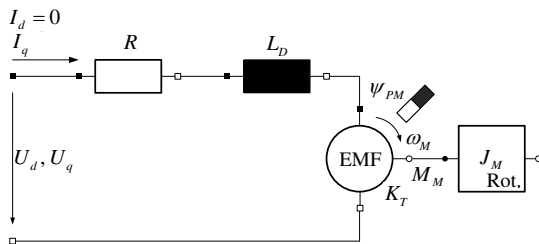


Figure 1: Equivalent circuit of the PSM

can be calculated from the given torque constant K_T by

$$\psi_{PM} = \frac{1}{3 \cdot p} \cdot K_T \cdot \quad (1)$$

The temperature of the motor T_M is dependent on the power dissipation P_V , the heat transfer resistance R_{Th} , the thermal time constant t_{Th} and the environmental temperature T_U , i.e.

$$R_{Th} \cdot P_V = t_{Th} \cdot \frac{dT_M}{dt} + (T_M - T_U). \quad (5)$$

While t_{Th} is mostly listed in the supplier specification, there is usually no value for R_{Th} given. But R_{Th} can be calculated from the stall current at 100 K overtemperature $I_{0,100K}$:

$$R_{th} = \frac{100 \text{ K}}{3 \cdot R \cdot (I_{0,100K})^2}. \quad (6)$$

In addition to the winding resistance heat is dissipated due to iron losses and bearing friction at a rate approximately proportional to $\omega_M^{1.5}$. Overall the resulting motor losses are therefore

$$P_V = 3 \cdot R_{St} \cdot I_A^2 + k_R \cdot |\omega_M|^{1.5}, \quad (7)$$

where I_A denotes the effective value of the armature current that is equal to I_q if no field weakening is applied. The proportionality factor k_R for the iron and bearing friction losses can be estimated by the manufacturer's specifications of the motor current $I_{N,100K}$ for the rated torque $M_{N,100K}$ and the rated speed ω_N , i.e.

$$k_R = \frac{K_T \cdot I_{N,100K} - M_{N,100K}}{\sqrt{\omega_N}}. \quad (8)$$

Table 1. Typically available physical data for a PSM

Variable	Unit	Physical parameter
p	-	Number of pole pairs
K_T	Nm/A	Torque constant
K_E	Vs/rad	Voltage constant at 20° C
R	Ω	Winding resistance at 20° C
L_D	H	Effective inductivity
t_{mech}	s	Mechanical time constant
t_{Th}	s	Thermal time constant
J_M	kg m ²	Rotor inertia

The manufacturer specifications contain nominal values. However, some parameters can be determined more accurately when the actual state of the motor is taken into account. For example the resistance of the windings changes with temperature according to

$$R^* = R \cdot (1 + \alpha \cdot (T_M - 293,15 \text{ K})) \quad (9)$$

with α as the temperature coefficient of copper at 20 °C. The torque constant K_T is usually given for an overtemperature of 100 K, i.e. $K_T = M_{0,100K} / I_{0,100K}$. Since the manufacturer data usually contains the stall torque and current also for 60 K overtemperature, a factor for the temperature dependence can be estimated by

$$c_{kT,T} = 1 + \frac{K_T - M_{0,60K} / I_{0,60K}}{40 \text{ K} \cdot K_T} \cdot (T_M - 393 \text{ K}). \quad (10)$$

Due to saturation effects at high currents, the torque constant decreases at torques $M_M \geq 2 \cdot M_{0,60K}$ (Bosch Rexroth AG 2009; Siemens AG 2010). With the information on maximum torque and current the reduction factor $c_{kT,M}$ can be obtained by linear interpolation, so that

$$K_T^* = c_{kT,T} \cdot c_{kT,M} \cdot K_T. \quad (11)$$

2.1.2 Requirement Model

The requirement model of the PSM consists of the limit values and an adequate metric, see Table 2. For example, the line-to-line armature voltage U_{An} is limited by the maximum output voltage of the converter $U_{U,\max}$, i.e.

$$\sqrt{3} \cdot U_{An,\max} \leq U_{U,\max}. \quad (12)$$

Substituting (2) and (3) into (12) yields for steady state

$$\left(\frac{p \omega_M L_D M_M}{K_T^*} \right)^2 + \left(\frac{M_M R^*}{K_T^*} + \frac{\omega_M K_T^*}{3} \right)^2 \leq \frac{U_{U,\max}^2}{3}, \quad (13)$$

which allows to calculate the voltage limiting characteristic. These can then be compared to the torque-speed diagrams that are usually given by the motor supplier. Figure 2 shows a good correspondence between model and catalogue data regarding the voltage limiting characteristic; while Figure 3 shows the correspondence between the model and the manufacturers

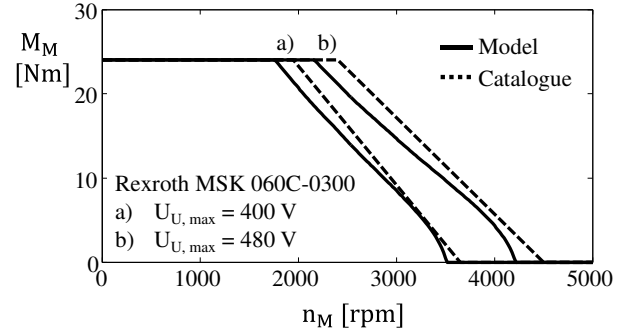


Figure 2: Comparison between calculated voltage limiting characteristics and supplier data sheet (Bosch Rexroth AG 2009)

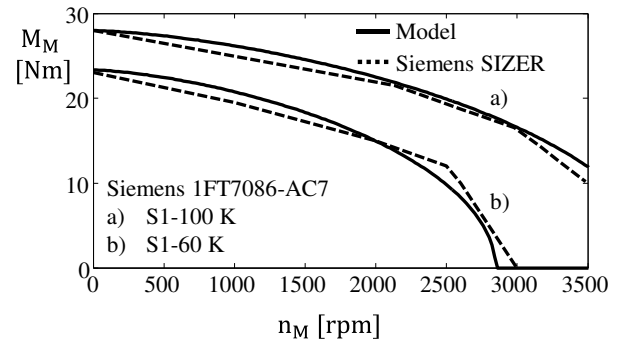


Figure 3: Comparison between calculated thermal limiting characteristics and supplier data sheet (Siemens AG 2010)

Table 2. Maximum permissible values for a PSM

Limit	Metric	Requirement
DC link voltage	$U_{An,\max} = \max_{t \in [t_0, t_f]} \left(\sqrt{(U_d(t))^2 + (U_q(t))^2} \right)$	$\sqrt{3} \cdot U_{An,\max} \leq U_{U,\max}$
Temperature limit	$\Delta T_{M,\max} = \max_{t \in [t_0, t_f]} (T_M(t) - T_U)$	$\Delta T_{M,\max} \leq \Delta T_{M,\text{perm}}$
Current limit	$I_{A,\max} = \max_{t \in [t_0, t_f]} (I_A(t))$	$I_{A,\max} \leq I_{M,\max}$
Torque limit	$M_{M,\max} = \max_{t \in [t_0, t_f]} (M_M(t))$	$M_{M,\max} \leq M_{M,J\max}$
Speed limit	$n_{M,\max} = \max_{t \in [t_0, t_f]} (n_M(t))$	$n_{M,\max} \leq n_{M,\text{perm}}$

sizing tool. However, for other motors larger deviations are observed, see Figure 4. The deviations can be explained by parametric uncertainties and nonlinearities in the interpolation.

To validate compliance with the temperature limit, the actual motor temperature is obtained by combining (5) and (7). Here, the manufacturer usually indicates the so-called S1-Curve in the torque-speed diagram. Operation points below the S1-curve allow steady-state operation without violating the temperature limit.

2.1.3 Modelica Model

The behavioral equations and the requirements of the PSM are included in a Modelica model that can be parameterized with typical manufacturer data, see Figure 5. Equations (2)-(4) are modeled with the electrical equivalent circuit consisting of inverter, resistance, inductance, air gap, zero potential and sensors for voltage and current. Note, that not the connectors from the Modelica Standard Library are used since both effective values in the dq-coordinate system have to be con-

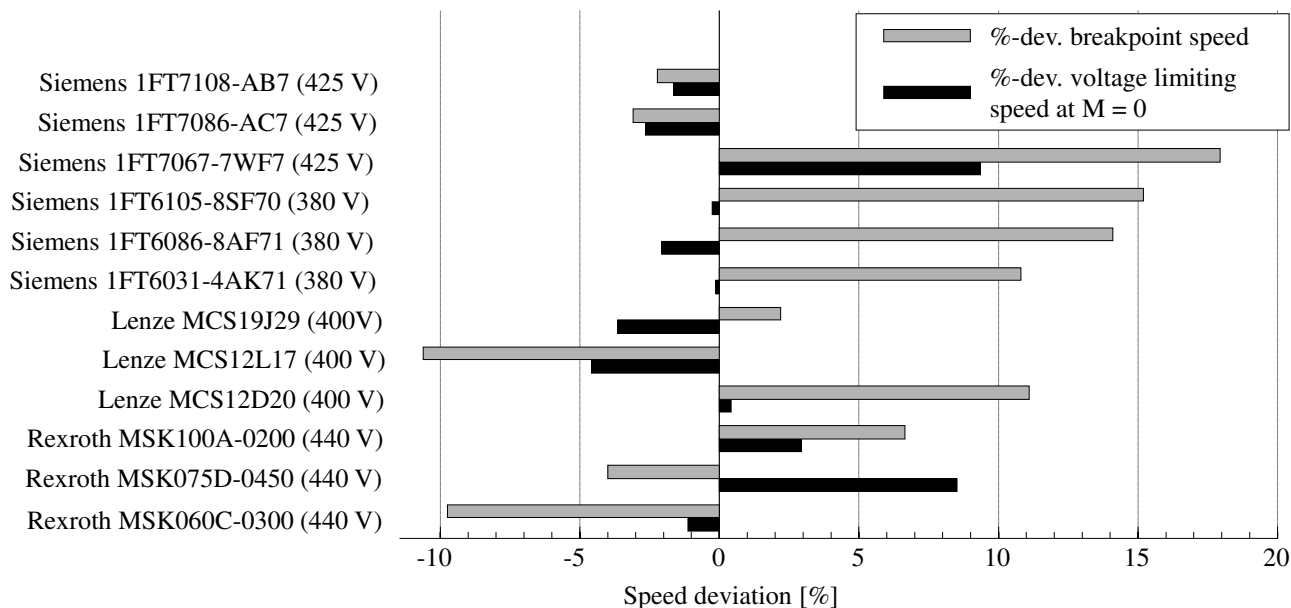


Figure 4: Percentage deviations of breakpoint speed and no-load voltage limiting speed for different motors

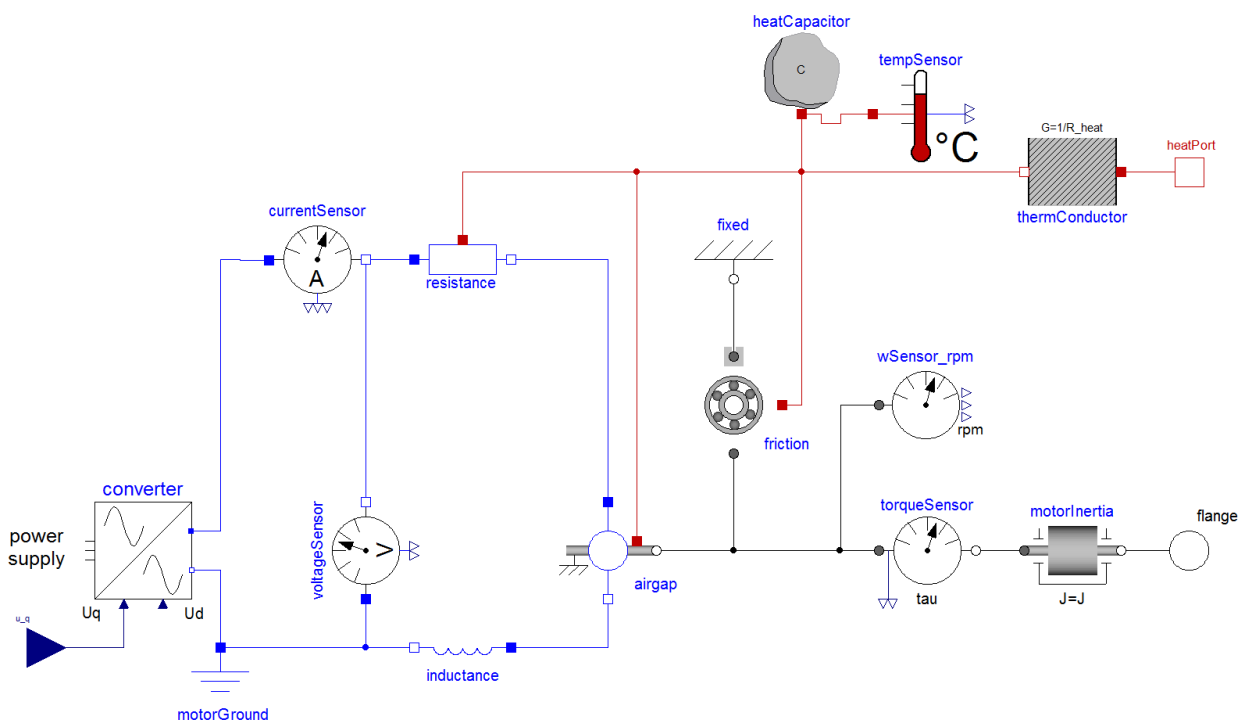


Figure 5: Permanently Excited Synchronous Motor in Modelica

sidered. This means the connector contains voltage and current in d- and q-direction. In addition, the electrical connector includes a fifth variable for the electrical angular velocity. In contrast to the motor models from the Standard Library the three phase current and the harmonic oscillations are not explicitly simulated, since these are not required to describe the control behavior in the context of machine tools so that the simulation time can be shortened.

With the model of the converter the effective motor voltage in d- and q-coordinate direction can be impressed. The flux-generating current I_d in the inverter is set to zero for the PSM without field weakening. The airgap model contains the relationship between torque-building current and torque (4) as well as between rotational speed and induced motor voltage (3). Moreover, the airgap includes the dependence of the torque constant on temperature and torque (11). The heat from the winding current and the iron and bearing friction losses yield the total heat loss that increases the temperature of the heat capacitor.

The metrics and requirements from Table 1 are included in the sensor models. For example, the current sensor contains the equations for maximum and RMS.

2.1.4 Validation of the Current Control Loop Model

According to (3), the relationship between torque-building current I_q and effective motor voltage in q-coordinate direction U_q can be described with a first order time lag. I_q is controlled with a PI controller and the electrical time constant is compensated by tuning the PI controller with the magnitude optimum criterion, see Figure 6 (left). The step response is measured for a servomotor of type Siemens 1FT6108 and compared to simulation, see Figure 6 (right). The settling time of simulation and measurement is similar, but the measured trajectory of I_q oscillates at a lower frequency. The frequency response – that has been obtained with Modelica Linear Systems Library – underlines this difference in the dynamic behavior since the maximum elevation of the simulation occurs at just under 200 Hz while the measured maximum lies at 600 Hz, see Figure 7. The amplitude response reflects the delays of the electrical system and the measurement system with a drop of 40 dB per decade. The differences between measurement and simulation are probably due to the simplified control structure in the model. Regarding the outer control loops in the cascaded feed drive structure, the difference can be neglected.

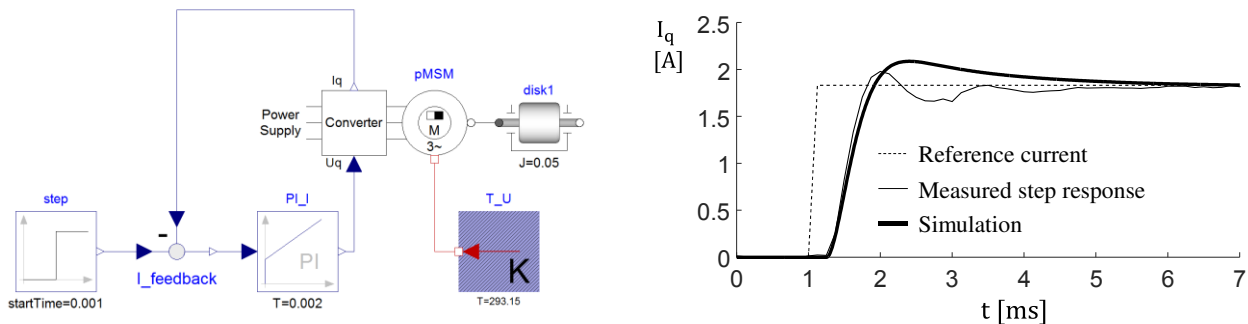


Figure 6: Model of the current control loop (left) and step response (right)

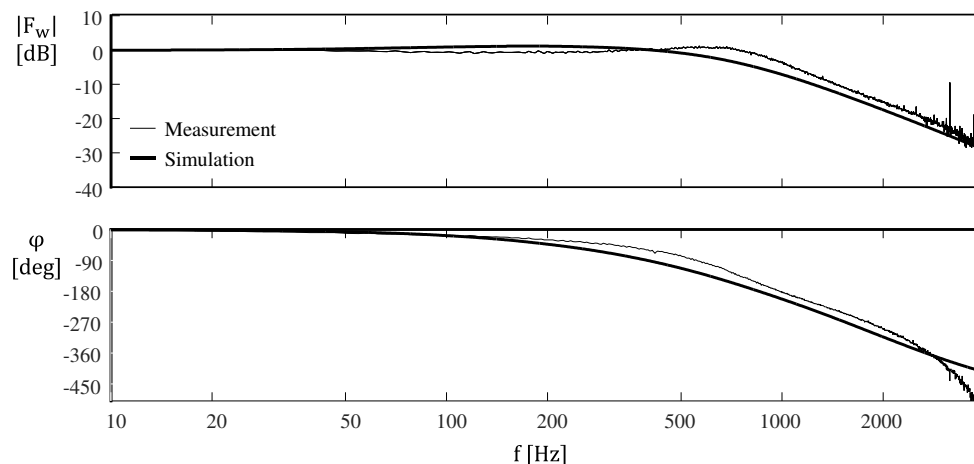


Figure 7: Reference frequency response of the current control loop

2.2 Model for Permanently Excited Synchronous Motor with Field Weakening Option (PSM-FW)

The aim of field weakening is to expand the possible operating range of the motor by eliminating the restriction of the voltage limiting curve. For this purpose a counter voltage is induced that weakens the magnetic field. For the PSM, field weakening can only be achieved by a current component in the d-coordinate axis that compensates the magnetic field of the permanent-magnet. Given the possibility of field weakening, but neglecting reluctance, (3) is extended to (Schröder 2009, p. 391):

$$U_q = L_D \frac{dI_q}{dt} + R^* \cdot I_q + p \cdot \omega_M (\psi_{PM} + L_D \cdot I_d), \quad (14)$$

which illustrates that a negative current I_d reduces voltage component U_q . On the one hand, the new limiting curve results from the condition $\sqrt{I_d^2 + I_q^2} \leq I_{M,\max}$. On the other hand, there is a new voltage limit at the point of maximum field weakening at $I_d = -\psi_{PM}/L_D$. In this case the product of rotational speed and torque is constant, i.e. the points on the voltage characteristic with field weakening correspond approximately to the same power.

From a modelling point of view, field weakening can be described as follows. If the converter output voltage without field weakening $U_{A,-FW}$ is below the permissible value, no field weakening is required. Once the converter output voltage without field weakening would exceed the permissible value, a negative current in d-coordinate direction is induced that maintains the voltage $U_A = \sqrt{3} \cdot \sqrt{U_d^2 + U_q^2}$ below the permissible value. Therefore, there are two alternative system equations:

$$\begin{cases} I_d = 0, & \text{if } U_{A,-FS} \cdot S_U \leq U_{U,\max} \\ U_A \cdot S_U = U_{U,\max}, & \text{if } U_{A,-FS} \cdot S_U > U_{U,\max} \end{cases}, \quad (15)$$

where S_U denotes a safety factor that determines how far the voltage is kept below the limit. In an exemplary case a motor is operated at a point just below the limiting curve of field weakening, see Figure 8 (left). The armature voltage increases with the speed up to the specified limit of $U_{U,\max}/S_U = 400$ V, see Figure 8 (right). Once the limit is reached, I_d keeps the voltage constant. The peak of I_d is just below the limit value of $-I_d = \psi_{PM}/L_D = 20$ A which corresponds to the operating point just below the field-weakening curve. It should be noted that an implementation according to (15) has the advantage that no controller tuning is required. However, reaching the maximum of the field weakening current leads to termination of the simulation since the algebraic equation system can no longer be resolved.

2.3 Model for Linear Motors

The percentage of linear induction machines as drive systems in machine tools is generally estimated below 10 %. Nevertheless it has its advantages in high dynamics and the lack of transmission elements. A disadvantage is the small accessible feed force in relation to the costs. The model of the linear motor follows from the PSM model by transferring rotational to translational dimensions, i.e. instead of ω_M we have translational velocity v_M , instead of p the pole pitch τ_p , and instead of K_T the force constant K_F . This yields:

$$U_d = -\frac{v_M \cdot \pi}{\tau_p} \cdot L_D \cdot I_q, \quad (16)$$

$$U_q = L_D \cdot \frac{dI_q}{dt} + R^* \cdot I_q + \frac{K_F^*}{3} \cdot v_M, \quad (17)$$

$$F_M = K_F^* \cdot I_q. \quad (18)$$

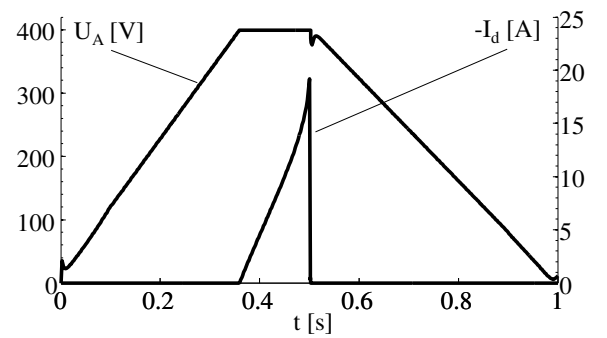
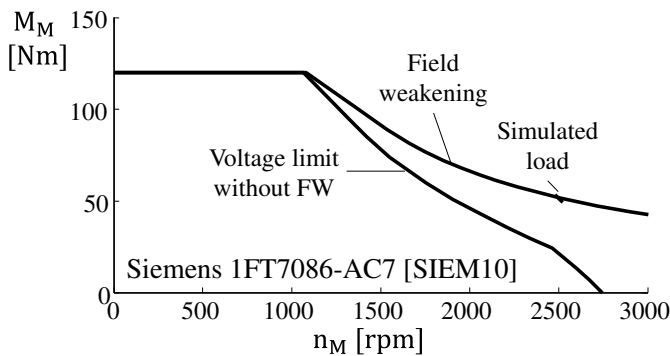


Figure 8: Operation points in the torque-speed-characteristic of the supplier (l), simulated trajectories for armature voltage and field-weakening current (r)

In terms of the thermal behavior, modeling of the linear motor varies from the PSM. Due to its structure a cooling system is deployed. The cooling cycle is described by

$$T_R = T_M + (T_V - T_M) \cdot \exp\left(-\frac{1}{R_{th} \cdot \dot{V} \cdot \rho \cdot c_p}\right), \quad (19)$$

where T_R is the recirculation temperature, T_V the forward flow temperature, R_{th} the thermal resistance, \dot{V} the volume flow, ρ the fluid density and c_p the spe-

cific heat capacity of the fluid. The pressure drop is obtained from the formula of Blasius

$$\Delta p = K_p \cdot \dot{V}^{1.75}, \quad (20)$$

where the factor K_p can be calculated from nominal flow rate and pressure drop as given in the manufacturer data. Figure 9 shows the linear induction motor model. The similarity to the rotational induction machine is obvious. The rotational parts such as rotor, rotational sensor and electromagnetic force have been

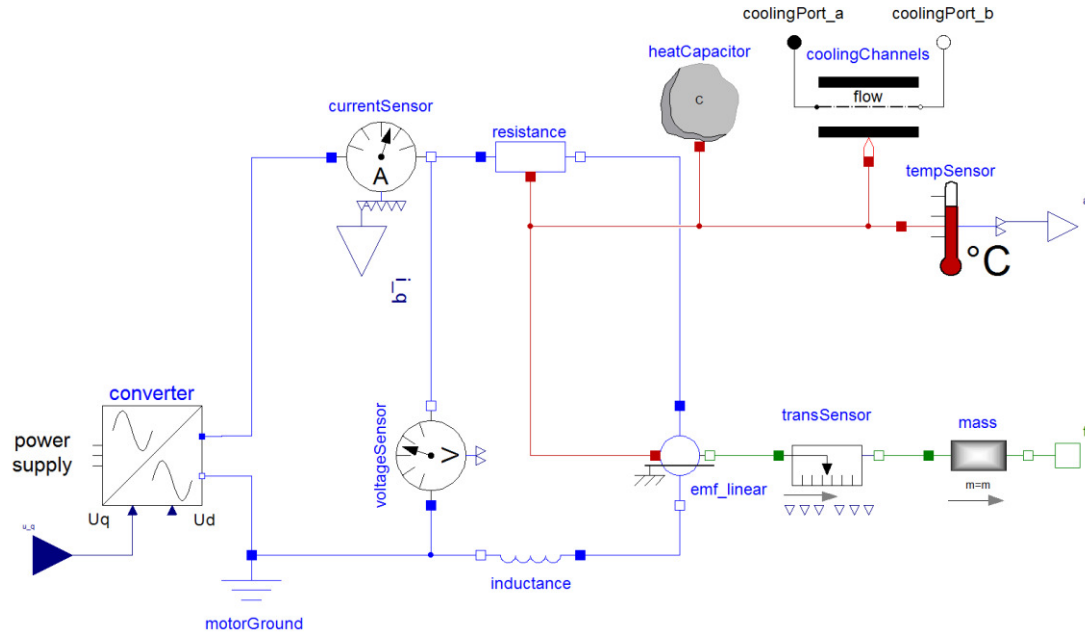


Figure 9: Modelica model for linear motors in Modelica

Table 3. Maximum permissible values for ball screw drives based on (Gross et al. 2006)

Limit	Metric	Requirement
Eigenfrequency	$f_d = \frac{1}{2\pi} \sqrt{\frac{1/m_{ref}}{1/c_M + l_{Sp}/(c'_{Sp} \cdot k_{Sp,L})}}$	$f_d \geq f_{d,min}$
Strain to preload	$F_{Sp,max} = \max_{t \in [t_0, t_f]} (F_{Sp}(t))$	$F_{Sp,max} \leq 2^{3/2} \cdot F_{a,VM}$
Strain to collapse load	$F_{Sp,max} = \max_{t \in [t_0, t_f]} (F_{Sp}(t))$	$S_{Kn} \cdot F_{Sp,max} \leq k_{Sp,Kn} \cdot \frac{d_{Sp}^4}{l_{Sp}^2}$
Strain to static rating	$F_{Sp,max} = \max_{t \in [t_0, t_f]} (F_{Sp}(t))$	$S_{0am} \cdot F_{Sp,max} \leq C_{0am}$
Critical bending speed	$n_{Sp,max} = \max_{t \in [t_0, t_f]} (n_{Sp}(t))$	$S_n \cdot n_{Sp,max} \leq k_{Sp,n} \cdot \frac{d_{Sp}}{l_{Sp}}$
DN-Value	$n_{Sp,max} = \max_{t \in [t_0, t_f]} (n_{Sp}(t))$	$n_{Sp,max} \leq DN_{perm} / d_{Sp}$
Lifetime	$\frac{L_h}{h} = \frac{2 \cdot (C_{am}/F_{ma})^3 \cdot 10^6}{n_m / \text{min}^{-1} \cdot 60}$	$L_h \geq L_{h,min}$

replaced by translational components. The electrical parts e.g. inverter and stator remain the same.

3 Mechanical Transmission Elements

Beside the servomotor, feed drives require transmission elements to transfer forces and moments from drive to tool. These elements also need a proper model to determine their influence on the drive system. Realistic component models can help predicting lifetime and strain limits of components. Taking inherent or geometric restrictions into account during simulation, the search for optimal system components is facilitated.

In the following the approach will be demonstrated by the example of a ball screw drive. Ball screw drives are the most frequently used feed elements for machine tools up to 4 m traveling distance to realize the transmission from rotational to translational motion. The translational motion is characterized by the spindle pitch h_{Sp} and the relation between translational and rotation velocity is given by

$$\frac{v}{\omega} = \frac{h_{Sp}}{2\pi}, \quad (21)$$

Spindle torque M_{Sp} and force F_{Sp} are described by a regular mechanical equation of motion

$$M_{Sp} = J_{Sp} \cdot \dot{\omega}_{Sp} + F_{Sp} \cdot \frac{h_{Sp}}{2\pi} + \sum M_F, \quad (22)$$

where $\sum M_F$ sums up the losses of the ball screw nut and the force dependent friction of the spindle bearings. The model contains the components spindle shaft, spindle bearing, spindle inertia and ball screw nut. The model for the screw nut defines the restrictions according to eigenfrequency, maximum strain, revolution

limit, lifetime and the lossy transmission from rotational to translational movement. The corresponding requirements in Table 3 are formulated similarly to those of the PSM.

4 Feed Drive System Simulation

A major advantage of using object-orientated models for feed drives is that modelling and simulation is possible with limited expert knowledge if an adequate component library is available. Once the system topology is known, the component models from the library can be connected and parametrized by the user from available data of catalogues. As an example for linked models one axis of a machine tool feed drive is shown in Figure 10. The design is parametrized by characteristic values that are available at an early design stage like stiffness of motor, clutch, spindle, spindle nut or bearings. Another advantage of the objective oriented approach is the intuitive connection of components. While signal oriented modeling demands mathematical knowledge regarding transfer from the user, the object-oriented approach allows to maintain the physical topology. In addition to the motor and the mechanical components Figure 10 shows the control loops. Using the Linear Systems Library one is able to determine the frequency response functions of the system, see Figure 11. As expected the correspondence between simulation and experiment shows differences in the dynamic behavior. This circumstance is owed to the simple model neglecting several compliances of the complex mechanical system. Nevertheless the dominant resonance at approximately 350 Hz is reproduced well by the model that only contains a-priori data. The simulation results show that with small modeling effort and a-

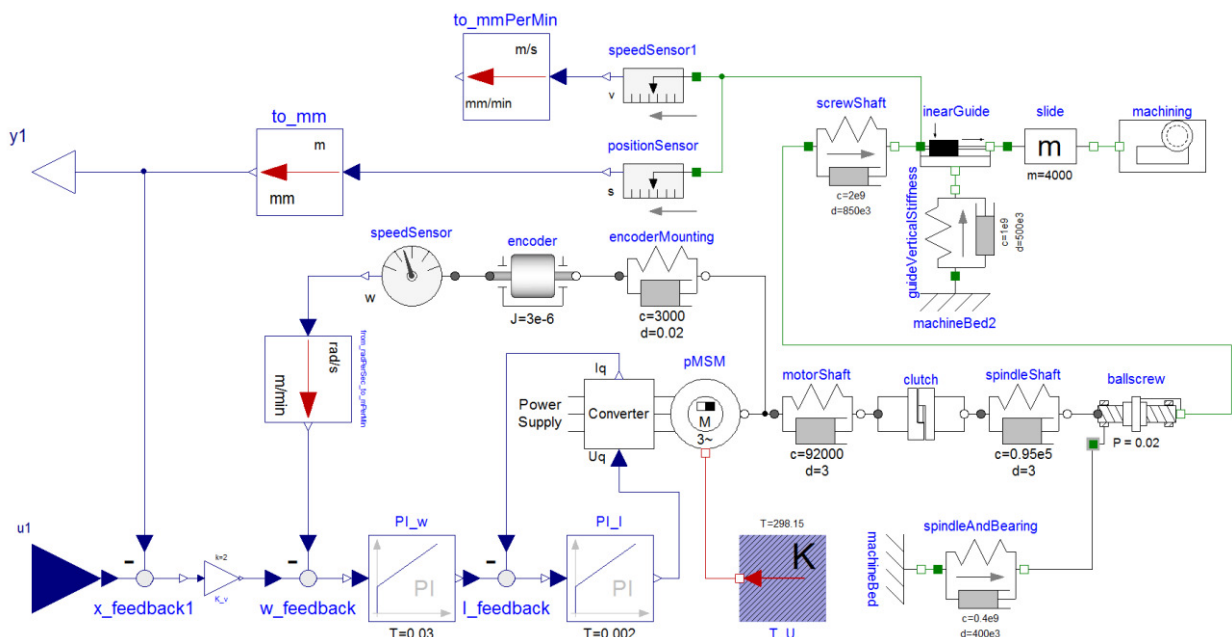


Figure 10: Drive topology for on axis of a machining center

priori available component data a prediction of the dynamic behavior and critical operation states is possible to some extent. For a detailed structural analysis, however, a FEM-simulation is necessary. Linked to a component database a sensitivity analysis can be deployed to find more suitable components in terms of the desired system properties like efficiency or other performance defining quantities.

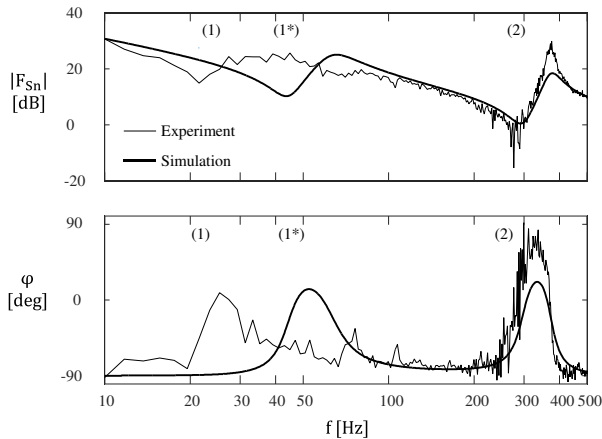


Figure 11: Correspondence between experimental and simulated frequency response

5 Conclusion and Outlook

This paper presented the Modelica library for feed drives. The behavior equations are based on well-known equations. It has been shown, how these equations can be incorporated in a model that can be parametrized with a-priori available data. Moreover, the paper outlines, how design requirements can be included in the component models. The library will be submitted to the Modelica Association as a free library by the end of 2015.

The final goal is to parametrize the object-oriented model so that it complies with the system specific restrictions such as limited dc voltage supply, temperature or energy consumption. The variables of the model serve as input for the linked optimization tool. Then the parameters of existing products, obtained from supplier catalogues for instance, are implemented in a database that is directly connected to the model library of the simulation tool. During an optimization run the optimizer has access to the database and is able to vary the model parameters at every iteration step. The optimization algorithm is therefore responsible for the systematic search for the best solution, without simulating the whole solution space that is containing all possible combinations of components. Beside the optimization of component parameters, it is also possible to analyze the system dynamics and to optimize the control system. By defining characteristic parameters for the dynamic behavior such as step response or frequency

response as target functions, the controllers are pre-designed directly during the engineering stage of the development process.

Acknowledgement

The authors thank the German Research Foundation DFG for the support within the project “Optimierung des Systementwurfs von Maschinen und Anlagen auf Basis komponentenorientierter Verhaltensmodelle”.

References

Altintas, Y.; Verl, A.; Brecher, C.; Uriarte, L.; Pritschow, G. (2011): Machine tool feed drives. In *CIRP Annals - Manufacturing Technology* 60 (2), pp. 779–796. DOI: 10.1016/j.cirp.2011.05.010.

Bosch Rexroth AG (2009): Rexroth IndraDyn S. Synchronmotoren MSK.

Brecher, Christian (2002): Vergleichende Analyse von Vorschubantrieben für Werkzeugmaschinen. RWTH Aachen: Dissertation.

Brecher, Christian; Brockmann, Birk; Daniels, Matthias; Wennemer, Matthias (2014): Herausforderungen bei der messtechnischen Untersuchung von Werkzeugmaschinen. In *Zeitschr. f. wirtsch. Fabrikbetrieb* (12), pp. 885–888.

Gross, Hans; Hamann, Jens; Wiegärtner, Georg (2006): Technik elektrischer Vorschubantriebe in der Fertigungs- und Automatisierungstechnik. Erlangen: Publicis.

Herfs, W.; Özdemir, D.; Lohse, W.; Brecher, C. (2015): Design of Feed Drives with Object-Oriented Behavior Models. In Inge Troch, Andreas Kugi, Felix Breitenacker (Eds.): *MATHMOD 2015 - 8th IFAC International Conference on Mathematical Modelling*, Vienna, pp. preprint.

Kral, C.; Haumer, A. (2005): Modelica libraries for dc machines, three phase and polyphase machines. In *Proc. of the 4th Modelica Conference*, pp. 549–558.

Kral, C.; Haumer, A. (2011): The New FundamentalWave Library for Modeling Rotating Electrical Three Phase Machines. In: *Proc. of the 8th Modelica Conference*, pp. 170-179

Özdemir, D.; Herfs, W.; Brecher, C. (2015): Approaching the Dilemma between Plan and Value in Computer Aided Engineering of Production Machines. In *Proc. of the 48th CIRP Conference on Manufacturing Systems, Ischia* accepted.

Schröder, Dierk (2009): Elektrische Antriebe - Grundlagen. 4th ed. Dordrecht: Springer.

Siemens AG (2010): SINAMICS S120. Synchronmotoren 1FT7. Projektierungshandbuch 03/2010. Erlangen: Siemens AG.

Simon, Walter (1986): Elektronische Vorschubantriebe an NC-Systemen. Technische Universität München: Dissertation.

Zirn, Oliver (2008): Machine Tool Analysis. Modelling, Simulation and control of Machine Tool Manipulators. ETH Zürich: Habilitation.

Model-based Development of a Holistic Thermal Management System for an Electric Car with a High Temperature Fuel Cell Range Extender

Dipl.-Ing. Torben Fischer¹ Dipl.-Ing. Florian Götz¹

Dr.-Ing. Lars Fredrik Berg¹ Dr.-Ing. Hans-Peter Kollmeier¹ Prof. Dr. rer. nat. Frank Gauterin²

¹Fraunhofer Institute for Chemical Technology (ICT), Project Group New Drive Systems, Germany
{torben.fischer, florian.goetz, larsfredrik.berg, hans-peter.kollmeier}@ict.fraunhofer.de

²Institute of Vehicle System Technology, KIT, Germany, frank.gauterin@fast.kit.edu

Abstract

Within the Fraunhofer innovation cluster “Regional Eco Mobility 2030” (REM2030) concept developments to improve the energy efficiency of regional eco mobility of the future are investigated. An AUDI A1 Sportback is used as a technology demonstrator with an entirely electric powertrain, completed to a serial hybrid by a fuel cell range extender. A methanol reformer provides hydrogen for the high temperature fuel cell. The main focus of this paper is the thermal management system of the car, which has to deal with different temperature levels and must be designed for zero emissions and energy efficiency. The model-based development of such a system using Modelica is described, comprising a conception, simulation and testing phase.

Keywords: REM2030; Thermal Management; Modelica; Electric Vehicle; Serial Hybrid; Heat Pump

1 Introduction and State of the Art

In conventional automobiles with combustion engines, waste heat is used to heat the passenger cabin. Electric cars do not dispose of sufficient waste heat to cover all calorific demands. Furthermore, additional components have a more sensitive operating temperature and require thermal conditioning. The thermal management system of the vehicle must therefore provide sufficient cooling and heating power. The range of an electric car is already limited due to the low energy density of the traction battery compared to conventional combustion fuels, and these thermal restrictions can cause a further range reduction. Using Modelica, a holistic thermal management system for an electric car, equipped with a high temperature fuel cell range extender, is developed. The goal is to minimize the electric energy demand of the thermal management system by using all heat sources and sinks in a holistic approach. In a further step the concept and the simulation will be validated on the basis of hardware testing. Once the functionality of the system is proven, additional

research will be dedicated to the operational strategy of the vehicle.

Thermal management systems for electrified powertrains can generally be classified according to the type of the battery cooling method and the heating technology applied. Battery electric vehicles often have a high-voltage PTC (positive temperature coefficient) heater to generate the demanded amount of heat (for example Tesla Model S, Smart Electric Drive and VW e-up). However, there are a number of vehicles which are optionally equipped with heat pumps instead of PTC heaters, like the VW e-Golf and the BMW I3. In the case of the Nissan Leaf II and Renault Zoe the heat pump is already integrated in series-production vehicles. The cooling media used to cool the battery are air, liquid and refrigerants, with refrigerants providing the highest efficiency but also showing the highest complexity. To the best of our knowledge, heat pumps as part of a holistic thermal management system are not yet commercially available, and are still the subject of research, for example in Germany within the projects GATE (“Ganzheitliches Thermomanagement im E-Fahrzeug”) and EFA 2014/2 (“Energieeffizientes Fahren 2014 Phase 2”).

2 Conception Phase

To develop a thermal management system, the system and its components must first be analyzed in terms of their nominal heat in- and output and their optimal thermal operating ranges.

2.1 System Description

In contrast to a conventional vehicle, the powertrain of an electric car includes the following components:

- Traction battery
- Power electronics
- Electric traction motor

The demonstrator vehicle of REM2030 is equipped with a 13 kWh Li-ion battery, assembled with 84 serial and two parallel cells. The traction motor is a permanently excited synchronous motor (PSM) and has a peak power of 80 kW, while the continuous power is at 50 kW.

The electrical range without accessory devices is projected to be about 80km, which can be increased by a range extender. The range extender is realized by a high temperature fuel cell with a methanol reformer (RMFC) providing an electrical power output of 5 kW (cf. Berg, 2015). With a tank volume of 11 liters, the range can be approximately doubled. The power electronics includes three DC/DC converters: a buck converter, which converts the intermediate circuit voltage (400V) to the onboard power supply (12V) and two boost converters, which convert the fuel cell voltage (150V) to the intermediate circuit voltage or the battery voltage (250V-350V) to the intermediate circuit voltage (400V). Finally, an inverter is installed to deliver AC voltage to the PSM depending on the requested torque and speed. This inverter can also be used to convert the AC charging current.

2.2 Heat Sources and Sinks

The operation temperatures of the components vary significantly, so they can act both as a heat source and a heat sink, depending on the ambient conditions. Additionally, the passenger cabin plays a major role in the thermal management system. Table 1 shows the identified heat sources and sinks within the system borders.

Component	Operating Temperature	Peak Heat Flow Rate
Battery	20-40°C	6 kW
Fuel Cell	140-180°C	5 kW
Cabin	20-25°C	-3..6 kW
Power Electronics	Up to 150°C	5 kW
Electric Motor	Up to 130°C	3 kW
Sun	.	1.5 kW

Table 1. Heat sources and sinks in an electrical vehicle

2.3 Efficient Heating and Cooling Concept

To achieve the defined goal it is necessary to use all heat sources to cover the heat demands. The residual heat demand must be covered at the expense of range, because the electrical energy of the traction battery is used. One promising approach to minimize electrical consumption for heat generation is to integrate ambient heat by using a heat pump. Cooling demands must also be covered using all the heat sinks. Usually, the

ambient air acts as a heat sink, but the temperature range of the battery and the cabin can be below the ambient temperature in some scenarios. An active cooling method must therefore be implemented. Conventionally, an AC-system is used, which can be achieved by a flow reversal in the heat pump. This is the state-of-the-art in reversible split air-conditioning units (Hundy et al., 2008).

Figure 1 shows the concept consisting of a heat pump with flow reversal, which is called a “thermal module”, and the connected periphery. The blue part on the right is the inner coolant circuit connected to the cabin heat exchanger, and the blue part on the left is the outer coolant circuit connected to the ambient heat exchanger. There are multiple operating possibilities of the thermal management system, which can be classified depending on the season. In the winter scenario the ambient temperature drops below the set temperature of the passenger cabin. If the range extender is running, the entire heat demand should be covered by the waste heat of the fuel cell. In extreme scenarios, the remaining heat demand can be covered by the heat pump. Waste heat from electric components will be released into the environment. In cases where the range extender is not running, the heat pump is turned on, while the waste heat of electric components is used as an additional heat source for the heat pump.

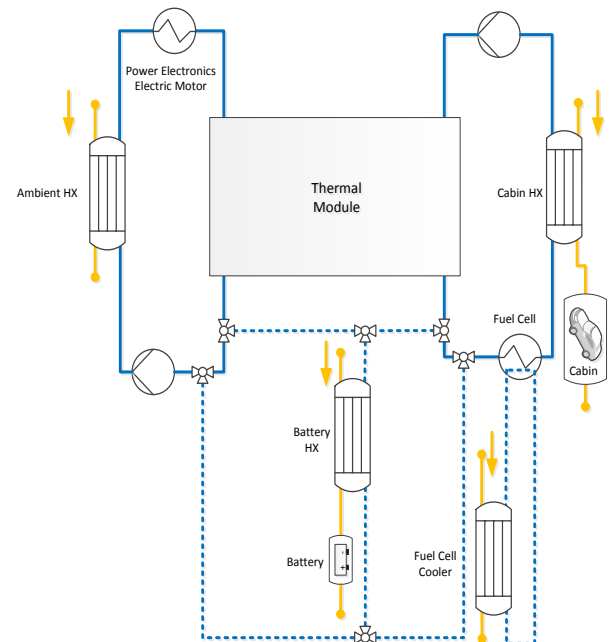


Figure 1. Thermal Management System

The summer scenario represents ambient temperatures above the passenger cabin set temperature. Waste heat from the electric components will be released to the environment, together with excessive heat from the cabin. The fuel cell has its own heat exchanger allowing the waste heat to be transferred into the

environment. The refrigeration process provides cooling energy to cool both the cabin and the battery.

In the transition phase the ambient temperature is considered to be slightly below the set temperature. This leads to two cases: Either there is too much waste heat from the fuel cell, which has to be partially released into the environment, or the range extender is not active, so that the heat pump has to overcome the remaining temperature difference.

The battery is air-cooled and can be connected to the coolant circuits via the battery heat exchanger. Since the battery needs to be heated and cooled, three-way-valves are installed, which allow the battery to be connected to the respective coolant circuit. The battery conditioning is therefore independent from the cabin air conditioning. This function could be necessary for example during a high demand phase on a cold winter day, when the battery has to be cooled while the cabin has to be heated.

3 Simulation Phase

To verify the function of the concept, a simulation model of the system was developed. Further potential for optimization might also be identified by running simulations, notably by carrying out a sensitivity analysis to identify the most significant parameters.

The models were built up in Modelica using the Dymola environment, based on the TIL Suite Library (Richter, 2008) by TLK Thermo GmbH and the Powertrain Library (Schweiger, 2005) by DLR. Thermodynamic fluid data are drawn from TILMedia and Refprop.

3.1 Thermal Module Model

The thermal module consists of two plate heat exchangers, two fixed orifice valves, a compressor, a four-way-valve and an accumulator. The heat exchangers change their function depending on the flow direction.

The plate heat exchanger model is based on a finite-volume-method and discretizes the flow path into three cells which each fulfill the energy and mass balance equations. Pressure drops within the heat exchangers are not taken into account. Heat transfer coefficients for the coolant side are based on the correlation for plate heat exchangers by the VDI (Martin, 2010), while the heat transfer coefficients for the refrigerant side are estimated on the basis of the Shah and Chen correlations (Shah, 1979 and Chen, 1966). The circuit runs on the refrigerant R1234yf, whose thermodynamic data are drawn from Refprop.

The 4-way-valve allows the flow direction to be adjusted, which is relevant for the initialization of the pressure states. The control of the valve is driven by the sign of the difference between ambient temperature and set temperature in the passenger cabin. The

initialization of the refrigeration system therefore depends directly on the ambient temperature. The receiver is placed on the low pressure side before the compressor, which is fixed in the flow reversal system. An inner heat exchanger, which is modeled by two tubes with a heat transport, ensures the best possible energy use.

3.2 Thermal Management Model

The thermal module is extended by two secondary coolant circuits, which release or receive heat from the connected components. The coolant flows through the converters and the electric motor, and they are modeled as tubes whose thermal input is calculated by the appropriate models in the complete vehicle system (see 3.4). The battery, fuel cell and passenger cabin are connected to the coolant circuits via fin and tube heat exchangers with a discretization of three cells. This is mainly due to a change in the heat transport medium from liquid to gas. The passenger cabin is modeled as a simple homogenous volume with a heat transfer to the surround environment. An air recirculation is implemented to quantify potential energy savings.

3.3 Battery Model

The battery is an important part of the thermal management system, as its temperature increases depending on the inner resistance and the current. The inner resistance, in turn, depends mainly on the temperature and the state of charge. The battery model therefore needs to include a feedback between electric output and thermal output, which influence each other.

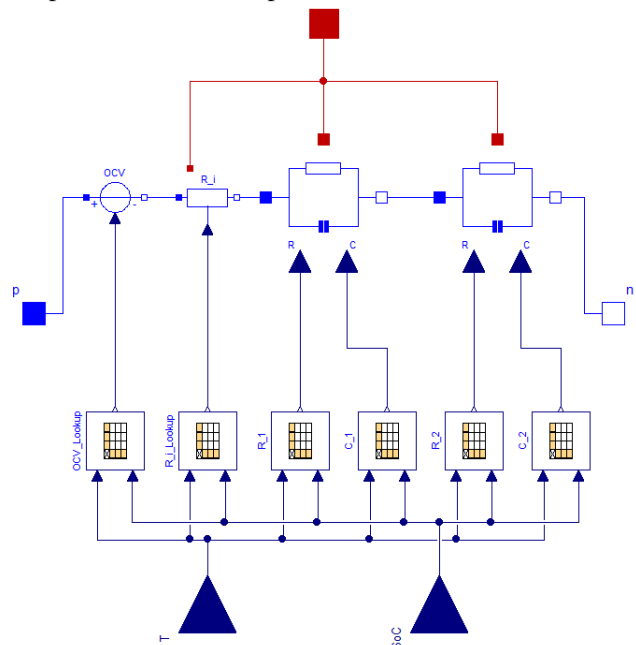


Figure 2. Equivalent circuit model with two RC elements

To develop a realistic, fast-running model, which should be used within a system simulation, an equivalent circuit diagram with an inner resistance and two RC elements was modeled. This method is widely

known and can be also found in the literature (Andre *et al.*, 2010).

Battery cycling tests deliver parameter values for the inner resistance, and $R1/C1$ or $R2/C2$ for the RC-elements at an open circuit voltage. These tests are performed within a climate chamber and cells are (dis-) charged with a given current, varying frequency, varying temperature and varying state of charge. Electrical impedance spectroscopy is applied to determine the battery parameters. A calorimetric measurement of the cell in a waterbath determines the specific heat capacity, which is needed to calculate the temperature of the battery.

The complete parametrized model has an electric-thermal feedback, in which the battery is modeled as a thermal mass and one cell is scaled to a whole battery pack.

Using the object-oriented capacities of Modelica, it is possible to instantiate one battery cell several times to build up a battery pack consisting of individual cells. This provides a good possibility to investigate cooling concepts on a component level, since the individual temperature of each cell can be calculated. However, this significantly increases the computation time, and is therefore not taken into account.

3.4 Complete Vehicle Model

The results from the thermal management system simulation show the total energy consumption of the compressor and the pumps. To evaluate the effect on the range, a complete vehicle simulation with longitudinal dynamics must be performed. The complete vehicle model is based on the Powertrain Library calibrated with our vehicle and component parameters, complemented by the battery and motor models developed.

An interface between the thermal management model and the vehicle model has been created, which encompasses the following considerations:

- Effects of the thermal management system on the battery temperature
- Influence of the electrical power input of the compressor on the state of charge (SOC) of the traction battery
- Heat dissipation of converters and motor

The map-based model of the electric motor provides a torque depending on the position of the accelerator pedal and the actual speed. The supplied torque is needed to overcome the driving resistance at constant speed. A surplus of torque results in increasing speed and thus in the higher velocity of the car. The motor model also delivers the efficiency rate at the actual operating point, which is used to calculate the amount of heat transferred into the coolant system.

To quantify the benefit of the thermal management system in terms of range and consumption, the WLTP

cycle (Worldwide harmonized Light vehicles Test Procedures) is chosen as a reference driving cycle.

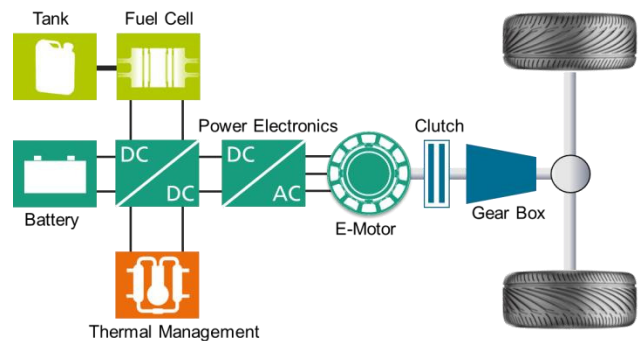


Figure 3. Complete vehicle system with electrical, mechanical or hydraulic connections

3.5 Simulation Setup

All simulations are carried out with the base vehicle outlined in the system description 2.1. The cabin volume is assumed to be 2.5m^3 , the overall heat coefficient of the cabin to the surrounding environment is defined to 40W/K according to measurements. The total window area is 2.8m^2 and the frontal area is 2.04m^2 with a drag coefficient of 0.33. The mass of the vehicle is defined to be 1200kg with a rolling resistance coefficient of 0.014.

4 Control System

4.1 Classic Control Approach

The thermal management system consists of several components, which must be maintained within their thermal operating range. This is either for comfort reasons, as in the case of the passenger cabin, or for life time and performance reasons, in the case of the battery and the fuel cell.

Relating to our system there are three set temperatures for different components:

- Passenger cabin ($\sim 22^\circ\text{C}$)
- Fuel cell ($\sim 160^\circ\text{C}$)
- Traction battery ($\sim 20\text{-}40^\circ\text{C}$)

In this multiple-input and multiple-output system (MIMO) every input influences more than one output. To apply a classic control approach it is therefore necessary to decouple the systems and to treat each partial system as a system with a single input and a single output (SISO). As stated in (Levine, 1999) it can be difficult or impossible to find a controller for a MIMO system, such that every input affects only one output. In this case, the compressor speed is controlled by the temperature in the passenger cabin, while the three-way-valve in the inner coolant loop is controlled by the battery temperature. If the battery needs to be cooled down, the 3-way-valve opens, which causes a temporary reduction of cooling power for the cabin.

The temperature change in the cabin eventually increases the compressor speed. So the control variable of the 3-way-valve influences both the battery and cabin temperature, but is still a simple approach to control this MIMO system.

However, this kind of control system always produces some overshoots, leading to an inefficient way of providing heating and cooling energy. Furthermore, the temperature change in the cabin might cause a feeling of discomfort for the passenger, which has to be avoided.

4.2 Dynamic Optimization

Another approach is to treat the whole thermal management system as a MIMO system. A cost function will be defined which includes the parameters to be minimized. As described in (Gräber et al, 2009), the system will be repeatedly solved online in an optimization loop, while the optimization algorithm changes the set of parameters and eventually keeps those resulting in a minimized cost function. This step is repeated in every given sample time for a given time horizon.

The computational time can be very costly, but since thermal systems react rather slowly, the sample time can be increased. This can lead to a more efficient control system, as overshoots can be avoided. Furthermore, it can be extended to a predictive control, which could eventually include data from a predictive operational strategy of the range extender.

The model developed can be used in its basic form to control the system using a dynamic optimization, although it has to be adapted when using gradient-based optimization algorithms. Look-up tables and fluid property models are particularly affected.

5 Preliminary Results

5.1 PTC vs. holistic approach with heat pump

First, the base car equipped with a state-of-the-art PTC heating element is compared to the base car with a heat pump system. The heating element is assumed to completely convert its electric energy to heat.

The results shown in figure 4 were simulated with an ambient temperature of 0°C, which is also the initial temperature of the cabin. The total consumption within the WLTP cycle and under the mentioned assumptions can be reduced from 18.3 kWh/100km to 14.3 kWh/100km, i.e. a decrease of about 22%.

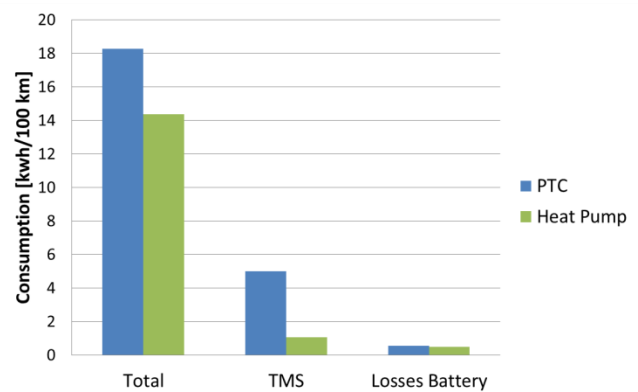


Figure 4. Comparison of the consumption of a conventional PTC heating element and a heat pump system at an ambient temperature of 0°C

Meanwhile, the electrical range can be increased from 60 to 75km. The coefficient of performance (COP), which is the coefficient between the thermal output and electrical input of the heat pump, is around 4.5. Pressure drops are not yet taken into account, as stated in 3.1. The battery cooling is not yet active, because its temperature rises from the initial temperature of 0°C to a maximum of only about 20°C by the end of the range.

5.2 Annual average advantage

Assuming that the air conditioning function of the thermal module provides a similar efficiency to that of a conventional air conditioning system, there will be neither an efficiency decrease nor an increase in the summer.

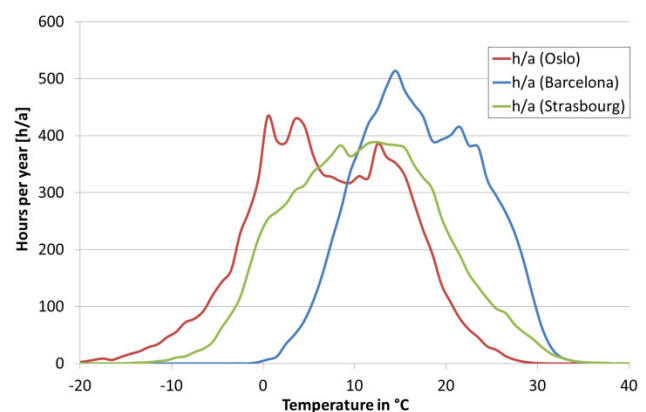


Figure 5. Average temperature profiles in different European cities (VDI 4710 Part 4 March 2014, Table B62, B99, B23 reproduced with the permission of the Verein Deutscher Ingenieure e.V.)

To calculate the annual average advantage of the developed thermal management system, the advantage as a function of the ambient temperature was determined. Using the VDI Norm 4710 (cf. Figure 5), which provides annual statistical meteorological data for European cities, it is possible to weight the

respective advantage according to the average annual amount of time during which it applies.

The results are shown in table 2, not taking into account any user behavior during different seasons or times of day.

City	Annual average consumption decrease
Oslo	15 %
Strasbourg	12.4 %
Barcelona	8.4 %

Table 2. Annual average consumption decrease using a holistic thermal management system rather than conventional electric heating

5.3 Extending Range with a High Temperature Fuel Cell

The methanol tank capacity is 11 liters, which equates to approximately 11 kWh of additional electric energy.

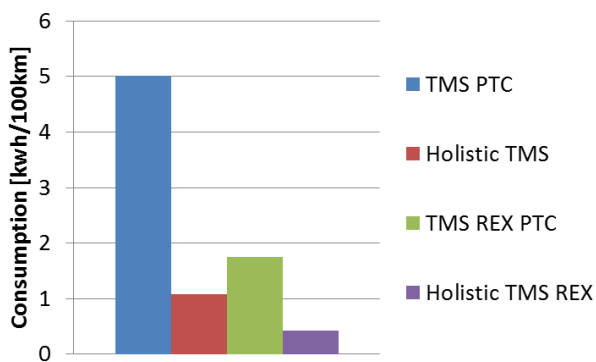


Figure 6. Comparison of the consumption of a conventional PTC heating element and a heat pump system at an ambient temperature of 0°C

With reference to the results of 5.1 the range can be extended to 143 km with a conventional heating element and to 153 km with the holistic thermal management system based on a heat pump. Here there is only a 7% advantage, because in both cases we use the waste heat of the fuel cell to heat the cabin: no additional heat has to be generated. Figure 6 illustrates this behavior. The average consumption of the thermal management system drops from about 5 kWh/100 km to 1.7 kWh/100 km when using a PTC element. By using a heat pump system this drop is from 1.1 to 0.4 kWh/100 km.

5.4 Preconditioning Battery

Another simulative investigation concerned the use of the preconditioning of a battery, which can be energy-intensive due to the high thermal mass of the battery. Since the battery's inner resistance is higher at low temperatures, it will heat up itself on sufficient power

demand. However, power output is limited at low temperatures, which might result in lower acceleration and also lower maximum velocity of the vehicle. Another issue is so-called lithium plating: the deposition of lithium on the anode as explained in (Korthauer, 2013). This leads to irreversible damage and occurs when charging the battery below 0°C.

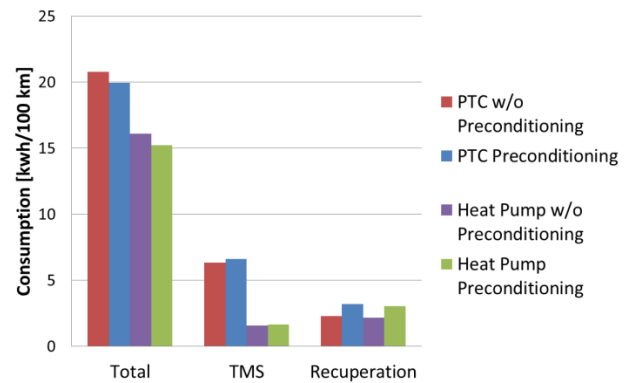


Figure 7. Comparison of consumption of a conventional PTC heating element and a heat pump system at an ambient temperature of -5°C

Taking this restriction into account by disabling the regenerative brake at temperatures below 0°C shows that preconditioning can decrease the total consumption. Figure 7 shows the simulation outputs with an ambient temperature of -5°C: Using a thermal management system with a PTC heating element, the total consumption can be decreased by about 4% by preconditioning the battery. While the amount of consumption of the thermal management system rises, the amount of recuperated energy rises more. The same effect can be observed using a heat pump system to precondition the battery, which leads to a decrease of the total consumption by nearly 6%.

6 Test Bench / Validation

A test bench for the heat pump has been developed and constructed to establish a validated, lossy model of the system including real heat transfers and pressure drops. The inner and outer coolant circuit can be thermally conditioned, allowing measurement at stationary temperatures. Several realistic scenarios, such as winter or summer conditions, can then be reproduced. The measurement includes temperature, pressure, mass flows and power signals while varying the compressor speed in different reproduced scenarios. This allows adaptation of the simulation model.

A dSpace system is used as a central control unit, which also has the opportunity to extend the test bench to a hardware-in-the-loop test bench. When testing the thermal management system, electric components such as motor and power electronics can be replaced by models, which calculate the output heat depending on

the driving cycle. Controlled heating elements then physically provide the calculated amount of heat.

7 Conclusion and Outlook

Modelica and the environment Dymola are powerful tools to support the development process in the early stage. With the help of these simulations, the potential advantages of the thermal management system for electric vehicles have been shown and the concept verified. A test bench has been constructed to adapt the simulation model of the refrigerant cycle to pressure drops and realistic heat transfer. The model-based development of the thermal management system eventually leads to the buildup in hardware, which will be connected to the components of the powertrain within the demonstrator. The model of the system can also be used to develop an advanced control system using dynamic optimization, which will replace the existing classical control approach. The functional mock-up interface (FMI), providing a simple possibility to exchange models between supported CAE-software, will play a major role within this undertaking. Finally, with a test on a dynamometer the functionality will be certified and the advantages of the systems will be proven.

M.M. Shah, "A general correlation for heat transfer during film condensation in pipes", *International Journal of Heat and Mass Transfer* 22, p.547, 1979

Verein Deutscher Ingenieure, VDI Norm 4710, Part 4 March 2014, Table B62, B99, B23

References

- D. Andre et al, "Characterization of high-power lithium-ion batteries by electrochemical impedance spectroscopy. II:Modelling", *Journal of Power Sources*, 2010
- L. Berg, "Range Extender mit Methanol-Fuel-Cell", HZwei, Hydrogeit Verlag, April 2015, p.30
- J.C. Chen, "A Correlation for Boiling Heat Transfer to Saturated Fluids in Convective Flows", *Ind. Eng. Chem. Process Design and Development* 5, p. 322, 1966
- M. Gräber et al., "Using Functional Mock-up Units for Nonlinear Model Predictive Control", 9th International Modelica Conference, Munich, 2009
- G. Hundy et al., "Refrigeration and Air-Conditioning", Butterworth-Heinemann, 2008
- A. Jeckel, „Thermische Anforderungen von Hochvoltbatterien in elektrischen Antriebssträngen“, VDI Wissensforum Thermomanagement für elektromotorisch angetriebene PKWs, 2013
- R. Korthauer, "Handbuch Lithium-Ionen-Batterien", Springer Vieweg, 2013, p.166
- W. Levine, "Control System Fundamentals", CRC Press, 1999, p. 166
- H. Martin, "Pressure Drop and Heat Transfer in Plate Heat Exchangers", "VDI Heat Atlas", Chapter N6, 2010
- C. Richter "Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems"
- C. Schweiger et al., "The PowerTrain Library: New Concepts and New Fields of Application", 4th International Modelica Conference, Hamburg, 2005

Predicting the Effect of Gearbox Preconditioning on Vehicle Efficiency

R. Gillot* A. Picarelli* M. Dempsey*

*Claytex Services Ltd. Edmund House, Rugby Road, Leamington Spa, CV32 6EL
{romain.gillot, alessandro.picarelli, mike.dempsey} @claytex.com

Abstract

Under extreme climatic conditions, the vehicle fuel consumption can be far from the certified value. Given the growing concern for polluting emissions, it is necessary to investigate a way to improve the overall vehicle efficiency and thus reduce the emissions and fuel consumption gap. One solution is to pre-warm the gearbox in order to make it work at an optimal temperature to achieve the best efficiency possible. Indeed, low lubricant temperature is a source of reduced vehicle efficiency due to the lubricant viscosity rising exponentially at very low temperature.

Using the Powertrain Dynamics library, a vehicle model with a detailed equation based gearbox model taking into account the temperature-dependent losses is developed.

Keywords: gearbox, pre-warming, efficiency, fuel consumption, oil temperature

1 Introduction

Responding to the ever growing need to reduce vehicle fuel consumption and pollutant emissions, new technologies have been developed and successfully implemented in a large number of vehicles over the last few years. However, if engine efficiency has recently dramatically increased thanks to ongoing design improvements and new technologies, the question is how much further we can push the limits to improve efficiency in use and at what price.

One way to achieve better performance from the powertrain is to improve its efficiency. To do so, we have to keep in mind that our vehicles are rarely operated in their optimal efficiency region due mainly to the road layouts, road traffic, driver behaviour, short range operation and the climatic conditions. We can at least seek to counteract the effects of the latter on vehicle efficiency. Vehicle transmission oil viscosity increases exponentially at low temperatures, affecting the vehicle transmission efficiency. Until the oil has fully warmed-up, which can take a rather long time under extreme cold weather conditions, the transmission losses are very high due to drag on the gears, clutches and bearings caused by the viscous oil. Poor range and fuel economy can result in customer dissatisfaction compounded by the fact that the vehicle is only being used exploiting a small percentage of its certified power. The idea is then to put the transmission (and in future other subsystems such as engine and traction battery as part of a larger study) in the best

conditions whatever the weather is in order to increase its efficiency. Farrant et al. showed the benefits of powertrain preconditioning during a cold start (Farrant P.E. *et al.* (2005).

In this paper we build a vehicle model in Dymola using components from the Powertrain Dynamics Library. We then precondition the transmission lubricant to several temperatures and run the vehicle model over the standard NEDC and ARTEMIS drive cycles. The ARTEMIS drive cycle combines an urban and a highway portion. The models involved in this study are predictive equation based models in order to show how the efficiency would benefit from higher oil temperatures without the constraints of map/empirical based models. The benefits of preconditioning are then highlighted as well as the costs of doing so.

2 The Vehicle Model

The vehicle includes a predictive thermal model of the transmission to quantify the thermal dynamics of the system including losses such as bearing and gear drag losses. The heat release from friction is evaluated in each area of the model. All gearbox components (moving and non-moving) are interlinked via mechanical and/or thermal ports so that the effects of each component in the system on the others can be evaluated. This physical relationship modelling forms the basis for predicting the oil temperature and viscosity in the whole subsystem.

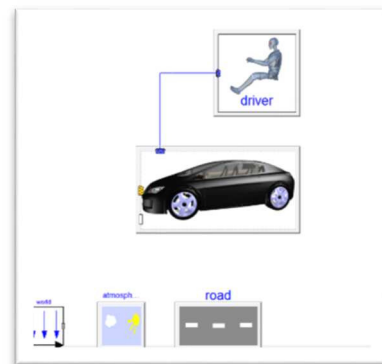


Figure 1. Vehicle model experiment complete with driver model, road and atmosphere (environment) models

The vehicle model is built on the VehicleInterfaces library standard from Modelica Association. All models use Multibody components and are designed for easy assembly and efficient computation (Dempsey M. *et al.* 2009). A Driver

model is included to exercise the vehicle model over a specific drive cycle in order to evaluate its efficiency and performance. A road model and an atmosphere model are also used in the experiment and interface with the vehicle model.

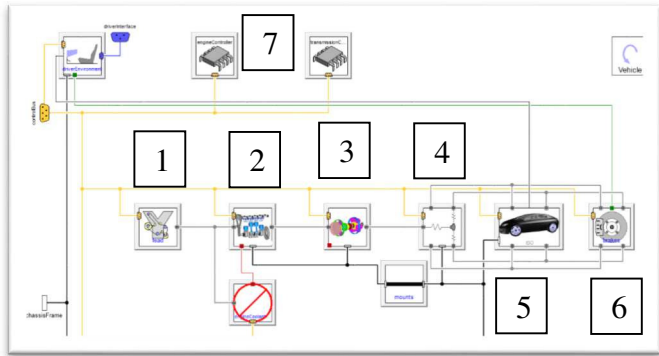


Figure 2. Detailed view of the vehicle model with all the subsystems:1-Ancillaries, 2-Engine, 3-Gearbox, 4-driveline, 5-Chassis, 6-Brakes, 7-Controllers

The model above (Figure 2) is a view of the vehicle one level lower than the previous one (Figure 1).

The main subsystems are: Engine, Transmission, Driveline and Chassis.

The engine model uses a performance map giving an output torque depending on pedal position and engine speed. A fuel tank is included to enable the calculation of the instantaneous and average fuel consumptions. The engine used in this experiment is a two-litre four-cylinder petrol engine.

The transmission model will be discussed in detail in the next section.

The driveline model can be set to include compliance, stiffness and damping characteristics so that the same vehicle can also be used for detailed driveability studies.

The chassis model has a pitch degree of freedom and longitudinal motion as well as models for tyres, aerodynamics and suspension.

3 The Transmission Model

3.1 Overview

In this study focus is on a six-speed automatic transmission which includes a dynamic torque converter model with predictive thermal effects, gear sets with bearing friction and bearing drag models, gear mesh models with temperature-dependent efficiency, a shift actuation system and a heat port (thermal connector) to port the generated heat to other vehicle subsystems.

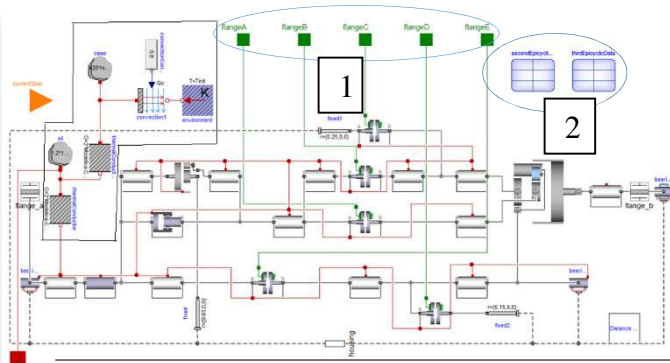


Figure 3. 6 speed automatic transmission gear set with thermal mass and heat dissipation network: 1-clutch actuation flanges, 2-data records for the epicyclic gear sets

The green connections in Figure 3 represent translational mechanical flanges to apply the clamp load to the clutches and brakes. The red connector is the heat port which us to thermally link all the components together in order to transfer the heat throughout the system and compute the temperatures at various points in the model.

The top left corner of Figure 3 is the thermal network for oil and casing. For each one of these components, a thermal conductor lies between the thermal masses representing the heat transfer to the oil and the casing respectively and the remainder of the gearbox’s thermal network. It is a systems modelling thermal model and we must also point out that the distribution of oil in the system is not considered in this paper.

3.2 Dynamic Torque Converter

In automatic transmissions, the torque converter couples the engine to the gearbox. Despite the availability of a steady-state (mapped) torque converted model, here we choose to use an equation based dynamic version since the transient response is of prime importance. There are three main components: the impeller connected to the engine, the turbine connected to the gearbox and the stator connected to the gearbox housing via a one-way clutch. The impeller is a centrifugal pump (Shin S., Bae I. *et al*, 2000, Jandasek V.J., 1994).

When the oil inside the torque converter is cold it affects the efficiency as the high viscosity decreases the impeller performance.

The moment-of-momentum equation is applied to the three control volumes. For example for the impeller:

$$I_i \dot{\omega}_i + \rho S_i \dot{Q} = -\rho \left(\omega_i R_i^2 + R_i \frac{Q}{A} \tan \alpha_i - \omega_s R_s^2 - R_s \frac{Q}{A} \tan \alpha_s \right) Q + \tau_i$$

ω_i is the impeller rotational speed, τ_i is the impeller torque, Q is the volume flow rate, I is the fluid inertia, ρ is the density, S is a design constant for the impeller, R is the impeller radius, A is the flow area and α_i is the impeller exit angle. We have similar equations for the turbine and stator.

The conservation of energy equation is:

$$\rho(S_i \dot{\omega}_i + S_t \dot{\omega}_t + S_s \dot{\omega}_s) + \frac{\rho L_f}{A} \dot{Q} = \rho(R_i^2 \omega_i^2 + R_t^2 \omega_t^2 + R_s^2 \omega_s^2 - R_s^2 \omega_i \omega_s - R_i^2 \omega_t \omega_i - R_t^2 \omega_s \omega_t) + \omega_i \frac{Q}{A} \rho(R_i \tan \alpha_i - R_s \tan \alpha_s) + \omega_t \frac{Q}{A} \rho(R_t \tan \alpha_t - R_i \tan \alpha_i) + \omega_s \frac{Q}{A} \rho(R_s \tan \alpha_s - R_t \tan \alpha_t) - p_L$$

p_L represents the losses, L_f is the fluid inertia length.

$$p_L = \frac{\rho}{2} \text{sgn}(Q) (C_{sh,i} V_{sh,i}^2 + C_{sh,t} V_{sh,t}^2 + C_{sh,s} V_{sh,s}^2) + \frac{\rho f}{2} \text{sgn}(Q) (V_i^{*2} + V_t^{*2} + V_s^{*2})$$

We also need to add a new term to the losses to correct the impeller efficiency:

$$p_{L2} = \frac{\tau_t \omega_t (C_n - 1)}{Q}$$

Where $C_{sh,i}$ is the shock loss coefficient for the impeller and $V_{sh,i}^2$ is the shock velocity. f is a fluid friction factor and V_i^{*2} is the fluid velocity relative to blades.

We know the fluid viscosity significantly affects the torque converter efficiency, this is why we need to include the thermal effects in the model.

A formula in (Hydraulic Institute, 2010) gives a performance factor (B) in order to calculate a correction factor to re-evaluate the impeller efficiency when using a viscous fluid.

$$B = K \left[\frac{V_{vis}^{0.5} H_{BEP-W}^{0.0625}}{Q_{BEP-H}^{0.375} N^{0.25}} \right]$$

Where V is the fluid viscosity, H is the impeller head, Q is the fluid flow rate and N is the impeller shaft speed. We can now use this performance parameter to determine the correction factors C_H for the head, C_q for the volume flow rate and C_n for the efficiency.

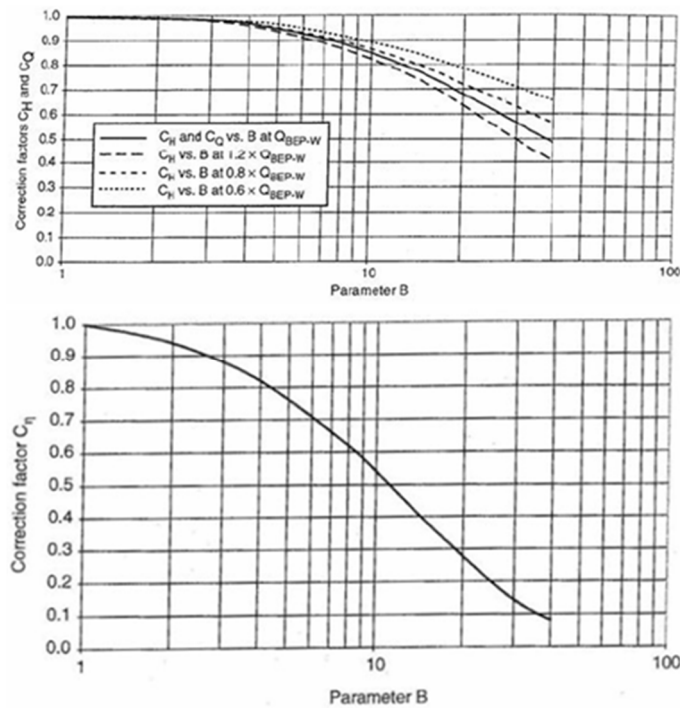


Figure 4. Correction factors C_H , C_Q and C_n plotted against the performance factor B.

These correction factors allow us to modify the losses to account for the thermal effects.

First we introduce the volume flow's corrected value:

$$p_{L,thermal\ effects} = \frac{\rho}{2} \text{sgn}(Q) (C_{sh,i} V_{sh,i}^2 + C_{sh,t} V_{sh,t}^2 + C_{sh,s} V_{sh,s}^2) + \frac{\rho f}{2} \text{sgn}(Q) \left(\frac{V_i^{*2} + V_t^{*2} + V_s^{*2}}{C_q^2} \right)$$

These corrections result in the turbine rotational speed to converge more slowly towards the impeller angular velocity as can be seen in Figure 5.

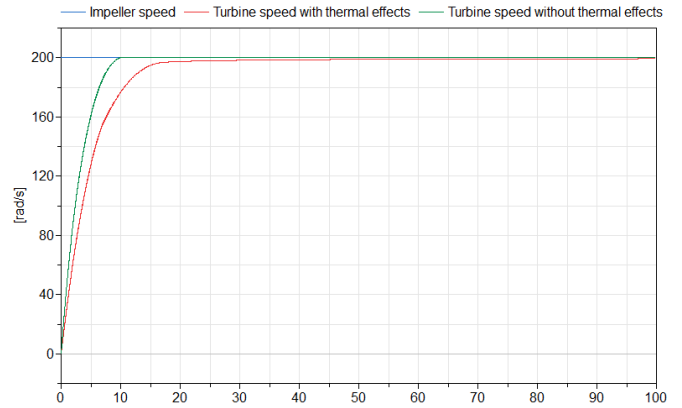


Figure 5. Impeller and turbine speeds vs. time [s] with and without taking into account the thermal effects.

When the oil is cold and has a high viscosity the flow's axial velocity inside the torque converter is reduced and so the fluid inertia transmitted to the turbine is not as good and it takes longer to reach the coupling point. The overall torque converter efficiency is then affected.

3.3 Roller bearings

The roller bearings are important components as they can achieve very good efficiencies under appropriate conditions. However, when the oil viscosity is high their efficiency dramatically drops. Since all the components in the transmission model are thermally linked, the heat released by other parts of the model (clutches, torque converter, etc.) can affect the bearing behaviour through warming up of the transmission fluid and gradually help to improve the overall system efficiency.

The bearing friction torque is given by:

$$T_{friction} = fn * coeff * R + T_{seals} + T_{drag}$$

The friction coefficient *coeff* depends on the type of rollers (ball, pin, taper pin, etc.) and typically varies between 0.001 and 0.0024.

The friction torque due to the seals is not detailed here as it has only a small contribution and does not vary significantly with temperature.

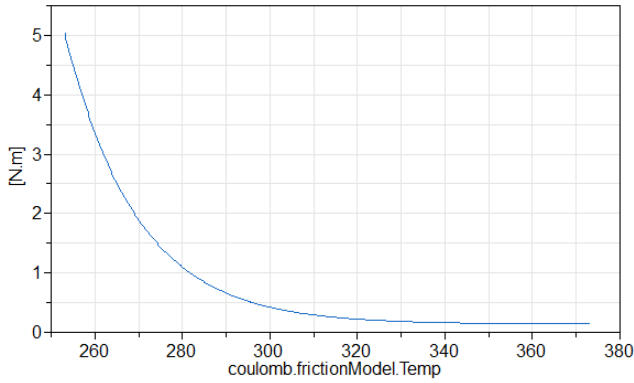
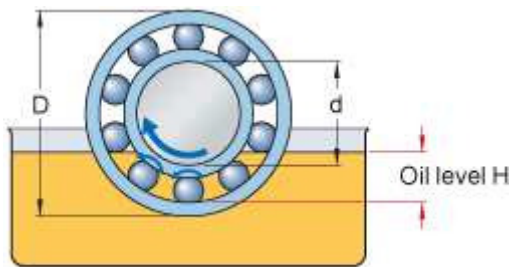


Figure 6: Friction torque of a roller bearing depending on the oil temperature

The drag torque formula (SKF website) demonstrates the importance of the oil viscosity:

$$T_{drag} = 4 * V_m * K_{roll} * C_\omega * B * d_m^4 * n^2 + 1.093 \times 10^{-7} * n^2 * d_m^3 * \left(\frac{n * d_m^2 * f_t}{\nu} \right)^{-1.379} * R_s$$

ν is the kinematic viscosity at operating temperature. C_ω and K_{roll} depend on the bearing geometry and dimensions. V_m is the drag loss factor. R_s is calculated with the bearing dimensions and the oil level (see picture below). n is the bearing rotational speed. f_t depends on the bearing geometry and the oil level.



The oil level is the distance above the lowest contact point between the rolling element and the outer ring.

3.4 Clutches

There are several clutches utilised throughout the transmission model. There is a lock-up clutch in the torque converter model and three other clutches are used in the gear set along with two brakes which use the same type of model. Clutches are modelled by multiple rotating plates pressed together via a normal force (via the green flange in the next picture).

The friction torque between the clutch plates is calculated as follows:

$$T_{friction} = \mu * N_f * (\varphi_f + \varphi_s) * \frac{\pi * w_{rel} * (r_o^4 - r_i^4)}{2h}$$

where μ is the oil viscosity, φ_f and φ_s are the pressure and shear stress flow factors respectively, introduced by Patir and

Cheng (1979). h is the oil film thickness which reduced under applied pressure (Dempsey M. et al. 2012). N_f is the number of friction plates.

The heat generated by the friction between the clutch plates is expelled through the heat port and stored into a heat capacitor which is linked to further thermal models in the system. The heat capacitor accounts for the thermal mass of driven and driving plates. The temperature of the clutch plates can thus be evaluated as well as the thermal losses to the clutch surroundings.

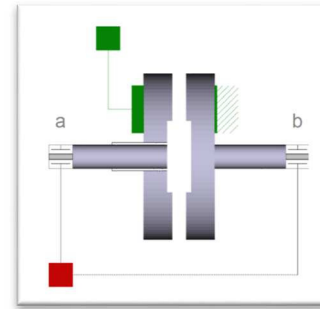


Figure 7. Clutch model with heat port (red) and mechanical actuation flange (green)

During slipping, clutches and brakes can generate considerable amounts of heat in the gearbox which contributes to lower the oil viscosity thus affecting the whole transmission efficiency. However, the friction torque is only significant for a small amount of time.

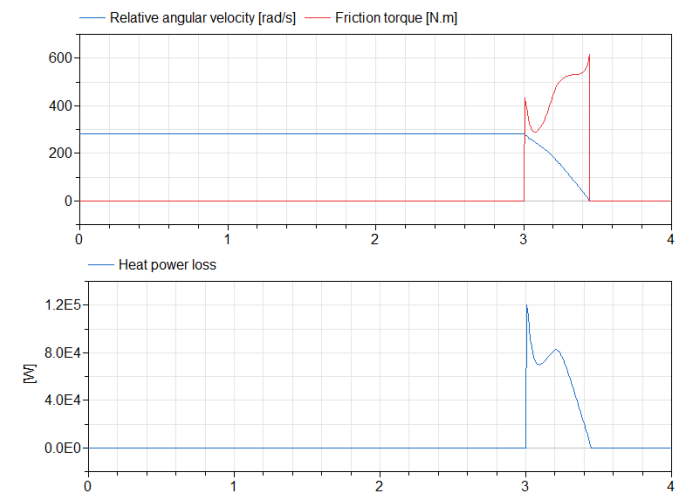


Figure 8. Friction torque and relative angular speed (top) and heat power loss (bottom) vs. time [s]

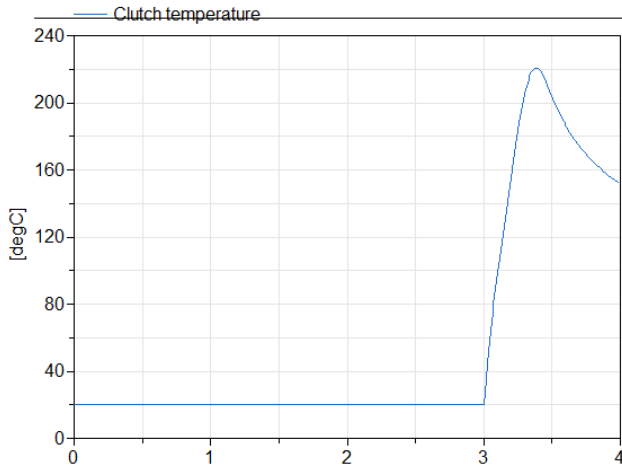


Figure 9. Evolution of the temperature at the clutch plates vs. time [s]

During operation, the clutch temperature can rise very quickly. We can see on the graph above (Figure 8) that heat is released when both the relative angular velocity and the torque transmitted are non-zero which essentially happens during the engagement and disengagement phase.

3.5 Gears

We can divide the gear losses into speed-dependent and load-dependent losses. Speed-dependent losses consist of windage losses and oil churning losses. Load-dependent losses consist of sliding friction losses and rolling friction losses.

The windage losses are induced by oil droplets that are in suspension in the gearbox housing and create a thin mist which increases the gear frictional resistance.

The churning losses are due to the oil being trapped in the gear mesh and to the gear rotating in the lubricant. They depend on the gear rotational speed, the oil viscosity, the gear geometry and the proportion of the gear submerged.

The sliding and rolling friction losses are dependent on the gear rotational speed and on the instantaneous coefficient of friction.

We are mostly interested in the churning losses since they are closely related to the oil properties.

The churning losses for tooth surface are given by (Heingartner P., Mba D., 2003):

$$P_{Cl} = \frac{7.37 f_{gi} * \mu * n_i^3 * D_i^{4.7} * b_i * \left(\frac{R_f}{\sqrt{\tan\beta}}\right)}{A_g 10^{23}}$$

f_{gi} is the gear dip factor that is to say the ratio of the gear dipping into oil. D is the outside diameter, β is the helix angle and R_f is the roughness factor.

The oil viscosity μ plays a major role in the amount of churning losses and thus we can easily see the importance of the temperature on these. n is the gear rotational speed and all the other parameters in the formula are geometrical dimensions.

Two other similar formulae exist to calculate the churning losses for smooth outside diameters and smooth sides of discs (i.e. shafts and gear side faces respectively).

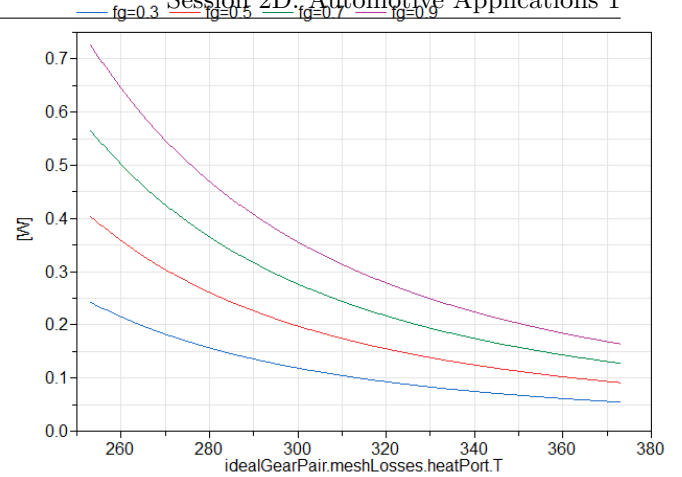


Figure 10: Example of oil churning losses with respect to oil temperature with different dip factors.

The churning losses increase at low temperatures and this is even a bigger difference when the dip factor is high as a larger part of the gear is dragged through oil.

4 Results

Prior to the calculation of vehicle fuel economy, a quick estimation of the cost to per-warm the gearbox can be made. If we apply a heat flow of 100 Watts, it takes 886 seconds to heat the gearbox from -10 degC up to + 90 degC. At an average price at the time of the study (in England) was £0.15 per kWh, the electricity needed will cost £0.0037. Even if the objective here is not to make money out of this solution, the cost has to be evaluated and the results clearly shows that it should not be a financial problem for the customer. This type of preconditioning is more suited in vehicles such as plug-in hybrids or EVs (Electric Vehicles) or indeed conventional vehicles operating in cold climates where the customer might, by default, plug the vehicle in to recharge the battery and/or precondition the cabin.

4.1 NEDC

The NEDC Cycle is used to homologate vehicles including Euro 6 standard. It is made up from an urban section repeated four times and an extra-urban section. It covers a distance of 11 023 meters and lasts for 1180 seconds. It is often criticised for not representing real-life driving conditions (too light duty). However it has to be considered as it is used for homologation but also because we are interested in slow urban driving conditions to attest the maximum savings possible (slower gearbox warm-up).

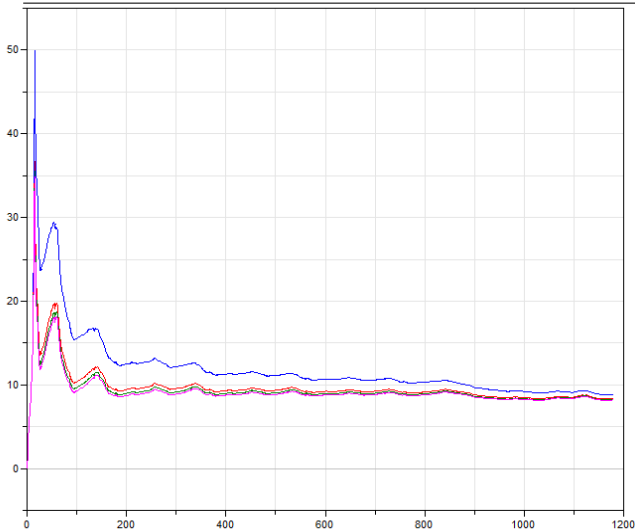


Figure 11. Average fuel consumption NEDC (l/100km) under different start temperatures: -10deg (blue), +23deg (red), +40deg (green), +90deg (magenta) vs. time [s]

We can see that the greatest saving occurs at the beginning of the cycle, in the urban portion at low load. While the oil temperature reaches its ideal value, the average fuel consumption with a pre-warmed gearbox converges towards the average fuel consumption with a non-pre-warmed gearbox.

If the ambient temperature is +23 degC, pre-warming the transmission fluid to +90 degC allows a fuel economy improvement of 1.71% (150 mL).

If the ambient temperature is now -10 degC, pre-warming the transmission fluid to +90 degC yields a fuel economy benefit of 7.66% (660 mL).

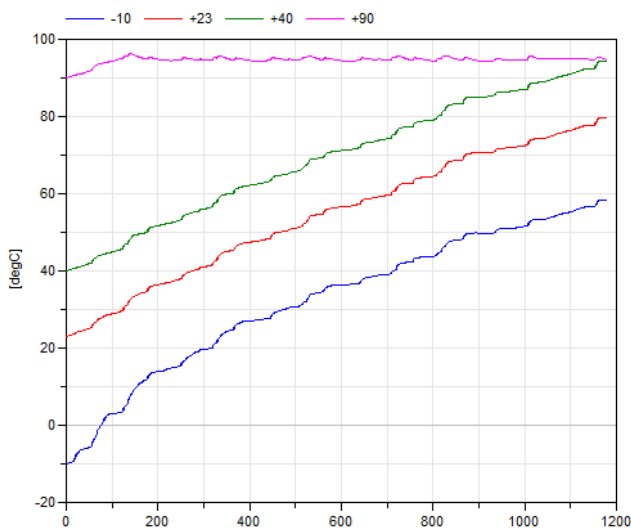


Figure 12. Oil temperature with different start values: -10deg (blue), +23deg (red), +40deg (green), +90deg (magenta) vs. time [s]

The oil temperature reaches its ideal value when starting at 40 degC only towards the end of the drive cycle while the others do not even come close, this explains the significant fuel savings.

The ARTEMIS drive cycle is based on a statistical study and thus fits better to the real usage of the vehicles. It is made of an urban part and a highway portion. It covers a distance of 33605 meters and lasts for 2061 seconds. In comparison with the NEDC drive cycle, ARTEMIS is much more aggressive.

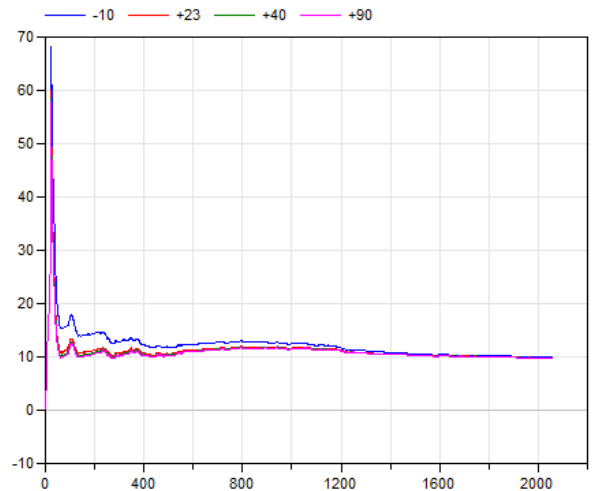


Figure 13. Average fuel consumption ARTEMIS (l/100km) under different start temperatures: -10deg (blue), +23deg (red), +40deg (green), +90deg (magenta) vs. time [s]

The potential fuel economy benefit from pre-warming the transmission is less obvious on this cycle. Indeed, as it is more aggressive, the components take less time to heat-up and so the benefit in pre-warming disappears more rapidly.

If the ambient temperature is +23 degC, pre-warming the transmission fluid to +90 degC yields a fuel economy improvement of 0.24% (37 mL).

If the ambient temperature is now -10 degC, pre-warming the transmission fluid to +90 degC yields a fuel economy benefit of 1.84% (182 mL).

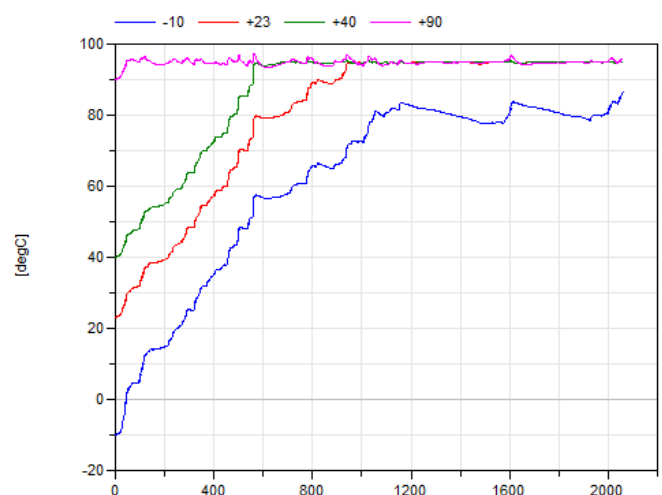


Figure 14. Oil temperature with different start values: -10deg (blue), +23deg (red), +40deg (green), +90deg (magenta) vs. time [s]

This graph (Figure 14) shows that the oil takes less time to reach its ideal value with the ARTEMIS drive cycle than with

the NDEC drive cycle. This is due to the ARTEMIS cycle being more aggressive.

An interesting occurrence that can be noticed in both drive cycles is that the overall vehicle efficiency does not increase much after the oil reaches +23 degC. The fuel economy when submitted to an initial temperature of +90 degC instead of +23 degC is only 1.71% for the NEDC cycle and 0.24% for the ARTEMIS cycle. This tells us that we could consider preconditioning the gearbox to only +23 degC as the benefits after this value are much less worth the cost.

4.3 Applied Example

Let's consider a real-life example to give us a concrete idea of the potential savings that could be achieved at a larger scale.

In Göteborg, Sweden (1 million people live in its urban area):

- 100 000 people commute to work every day
- 60 000 people commute by car every day
- 12 km (7.5 miles) / 24 minutes is the average commute ride inside Göteborg
- 51 km (31.7 miles) /50 minutes is the average commute ride outside Göteborg

If we make the assumption of 50 000 commuting trips per day, it gives us a saving a 25 000 litres of fuel and 62 tons of CO₂ per day.

5 Conclusions

The benefits of gearbox preconditioning have been studied and show that this strategy could be a real possibility in the future. A device to pre-warm the gearbox and the way for the customers to choose when to use it have still to be explored though.

As the differences in the results between the NEDC and ARTEMIS drive cycles showed us, the best potential savings could be achieved in a busy urban environment where the transmission fluid would usually take a long time to reach its ideal working temperature.

Pre-warming of other subsystems like traction batteries would allow even greater fuel economy and performance benefits and will be considered in further work. In the case of electric vehicles it is even a necessity to heat up the battery in extremely cold climates due to the performance degradation of the battery at very low temperatures. It has been shown (Tikhonov K., Koch V.R) that the battery discharge can reach 60% when the air temperature drops from 22 °C (72 °F) to -40 °C (-40 °F). To the evident issue when driving more than a couple of hours adds up the bad perception of the user if the engine can only deliver half of the 100bhp at cold start.

References

Dempsey M. et al. (2009) *Investigating the Multibody Dynamics of the Complete Powertrain System*, Como, Italy, Proceedings of the 7th Modelica Conference

Dempsey M. et al. (2006) *Coordinated automotive libraries for vehicle system modelling*, Vienna, Austria, Proceedings of the 5th International Modelica Conference

Farrant P.E. et al. (2005) *The Application of Thermal Modelling to an Engine and Transmission to Improve Fuel Consumption Following a Cold Start*, Toronto, Canada, Vehicle Thermal Management Systems Conference and Exhibition.

Heingartner P., Mba D., (2003) *Determining power losses in the helical gear mesh*, Chicago, United States, International Power Transmission and Gearing Conference.

Hydraulic Institute, (2010) *Effects of liquid viscosity on rotodynamic (centrifugal and vertical) pump performance*.

Jandasek V.J., (1994) *Design of Single-stage, Three-element Torque Converter, Design Practice: Passenger Car Automatic Transmissions*, Third Edition, AE-18, SAE, pp.75~102

Shin S., Bae I. et al, (2000) *The Effect of Blade Geometry on the Performance of an Automotive Torque Converter*, FISITA World Automotive Congress, Seoul, Korea

SKF website: <http://www.skf.com/group/products/bearings-units-housings/ball-bearings/principles/friction/skf-model/drag-losses/drag-losses-in-oil-bath/index.html>

Tikhonov K., Koch V.R., *Li-ion Battery Electrolytes Designed For a Wide Temperature Range*, Covalent Associates, Inc.

Model Based Development of Future Small Electric Vehicle by Modelica

Yutaka Hirano¹ Shintaro Inoue¹ Junya Ota¹

¹Toyota Motor Corporation, Japan, {yutaka_hirano, shintaro_inoue_aa, junya_ota}@mail.toyota.co.jp

Abstract

For future low carbon mobility society, new-type small electric vehicles (EVs) are developed actively in recent period. To reduce the energy consumption in various actual driving conditions, considering overall running resistance from tire characteristics, mechanical losses and electrical losses is necessary. In this paper, model-based development of system performance of a new EV is described. Full vehicle model considering both vehicle dynamics and energy consumption was developed using Modelica. Research for both structure and specification of components of the vehicle and also of the control were performed to find the solution to satisfy both energy consumption and vehicle dynamics by using the full vehicle model. Finally trade-off between vehicle stability and energy consumption and also between driver workload and energy consumption by using direct yaw moment control was indicated.

Keywords: Model Based System Development, Vehicle Dynamics, Energy Consumption, Electric Vehicle

1 Introduction

To satisfy needs for future low-carbon mobility society, development of many new small electric vehicles (EVs) is increasingly active in recent years. Those vehicles are often smaller and lighter than conventional vehicles and are often equipped with low RRC (Rolling Resistance Coefficients) tires for less energy consumption. On the other hand, low RRC tires tend to have less cornering performance than conventional tires in general. Because of light weight and low RRC tires, those vehicles become to have reduced dynamic stability against external disturbances such as side wind. To analyze and cope with all the problems about energy consumption and vehicle stability, a holistic approach of vehicle system design considering multi-physics of mechanics, electrics, aerodynamics, control and so on is necessary.

For this purpose, authors made an integrated model of the total vehicle system using acausal multi-domain physical modeling language Modelica (Hirano, 2014). By using Modelica, it is only necessary to define physical relationship written as equations in each component model and connect those component models as same as assembling the components to make the model of the whole system by hierarchical way.

This feature of Modelica is very powerful for model-based system development because it enables to modify the whole vehicle model very easily by using the results of experiments and physical investigations of each component. It is just necessary to replace the existing equations of the component model to the modified ones and replace the component model to the revised one by object-oriented way.

In the previous paper (Hirano, 2014), authors showed the capability of new construction of the new EV using new type tire based on ‘Large and Narrow concept’ and torque vectoring differential gear. For the model based development of the new EV, various kind of running resistance, vehicle dynamic performance and proper design of electric regeneration system were studied.

In this paper, a multi-physics full vehicle model of the new EV is expanded to consider the detailed loss of motors and inverters. Also front and rear suspension model which has same 3D mechanical design as the real experimental vehicle was made and verified. By technical investigations using this full vehicle model, structure, specifications and control of the new EV system were researched about vehicle dynamics and energy consumption.

2 Characteristics of Target EV

Table 1. Specifications of new experimental EV

	New EV	Conventional car
Vehicle Weight	750 kg	1240 kg
Yaw Moment Inertia	869 kgm ²	2104 kgm ²
Wheelbase	2.6 m	2.6 m
Front : Rear Weight Distribution	0.48 : 0.52	0.62 : 0.38
Height of CG	0.38 m	0.55 m
Aerodynamic Drag × Frontal Area	0.392 m ²	0.644 m ²
Tire RRC	5×10^{-3}	8.8×10^{-3}
Tire Normalized CP	16.1	20.4

The proposed experimental EV has specifications as shown in Table 1. Compared with a conventional small-class passenger car, the new EV has characteristics of lighter vehicle weight, smaller yaw moment of inertia, lower height of the center of gravity

(CG) and lower RRC value of tires. Because of these characteristics, this new EV is expected to have better handling and lower energy consumption than conventional vehicles. On the other hand, because of lighter weight and lower value of tire normalized CP (Cornering Power), this new EV seems more sensitive against external disturbances such as crosswind and road irregularity than the conventional cars. To cope with this problem, direct yaw moment control (DYC) was applied by using a new integrated transaxle unit for rear axle which has a main electric motor and also torque-vectoring differential (TVD) gear unit with a control motor.

3 Full Vehicle Model

3.1 Structure of the Full-Vehicle Model

To consider total balance of energy consumption, handling, stability, ride comfort and NVH (noise, vibration, harshness) of the vehicle, a full vehicle model including mechanics, electronics, vehicle dynamics and control was developed based on commercially available Vehicle Dynamics Library (Modelon, 2014). The vehicle dynamics model was built as a full 3-dimensional (3D) multi-body-dynamics model of all of vehicle body, suspension, tires and power train. Aerodynamics was also considered in the vehicle dynamics model. Component models of control systems such as TVD gearbox, electric motor and inverter were newly developed and connected with the full vehicle model. The control logic for DYC was also implemented as a controller block model. Additionally, driving environment such as road shape, side wind and air parameters (density, temperature, etc.) can be defined as the environment model. Figure 1 shows the top level of model hierarchy of the full vehicle model.

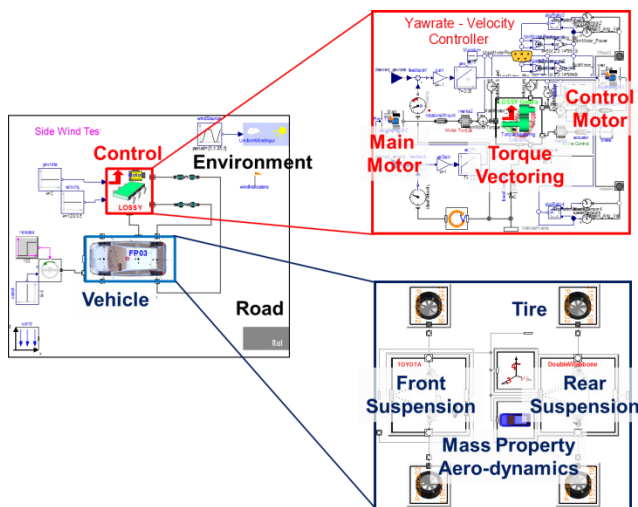


Figure 1. Top level structure of full vehicle mode

3.2 Mechanical Power by Driving Resistances

Power consumption of each system was calculated simultaneously to investigate the good balance of energy consumption and vehicle performances. At first, total mechanical power of driving resistances acting on the vehicle was calculated by following equations (Kobayashi et al., 2013).

Total driving resistance power:

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

Rolling resistance power:

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

Aerodynamic resistance power:

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

Cornering drag resistance power:

$$P_{sy} = \left[\left(\frac{d_f}{C_{pf}} + \frac{d_r}{C_{pr}} \right) M A_y^2 / g \right] \times V \quad (4)$$

Longitudinal resistance power:

$$P_{sx} = (M A_x + M g \sin \theta) \times V \quad (5)$$

Here

μ_r : tire rolling resistance coefficient (RRC),

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

M : vehicle mass [kg],

V : vehicle speed [m/s],

ρ : 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],

θ : road inclination [rad].

Total mechanical power of driving resistances can be calculated by equation (1) to equation (5) by using state variables of vehicle motion.

3.3 TVD Gear Train Model

For the TVD gear train, a driveline structure referencing the MUTE project of the Technische Universität München (TUM) (Höhn et al., 2013) was selected and the TVD model was constructed using commercially available Power Train Library (DLR, 2013).

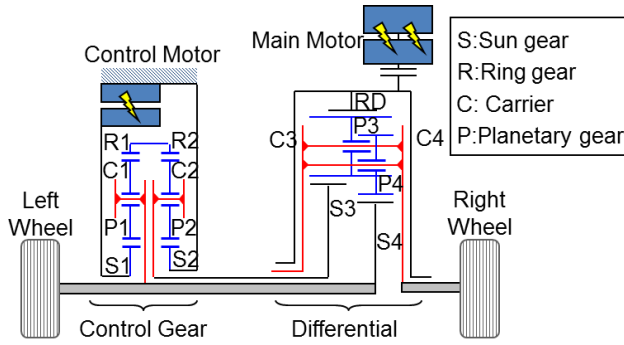


Figure 2. Torque vectoring differential (TVD) driveline

Figure 2 shows the configuration of the gear trains. This gear trains have a complex configuration constructed from several planetary gear sets. Torque from the main motor is distributed equally to the left and right wheel through the differential gear. Sun gear 3 is connected to carrier 2 in the control gear portion, and this configuration generates differences in torque between the left and right wheel by increasing or decreasing the torque distributed to the wheel on one side by torque input of the control motor. Specifications of the motors are shown in Table 2.

Table 2. Motor specification

	Main Motor	Control Motor
Max Torque	65 Nm	40 Nm
Max Speed	10,000 rpm	1,050 rpm
Max Power	15 kW	2 kW

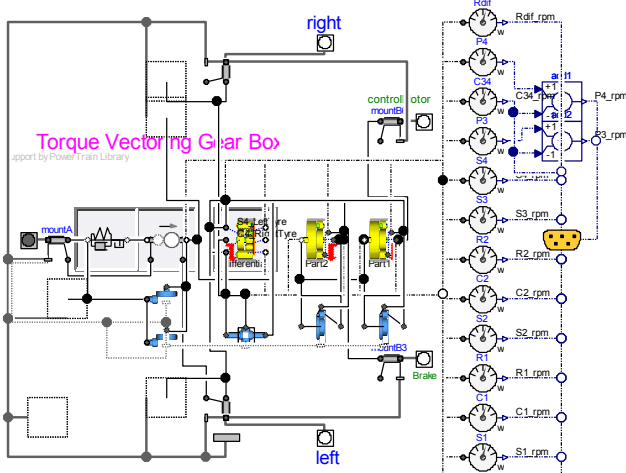


Figure 3. Modelica model of TVD gear train

Figure 3 shows a diagram of Modelica model of the torque vectoring gear train. The model is provided with elements that define the relational expression between the torque and speed of each gear engagement portion. Furthermore, these elements are capable of factoring in the overall gear torque loss by defining the torque loss for each element by following equations[7].

$$\Delta\tau = \begin{cases} (1-0.97)\tau_A & (\omega_{AB} > 0, \tau_A \geq 0) \\ (1-1/0.97)\tau_A & (\omega_{AB} > 0, \tau_A < 0) \\ (1-0.97)\tau_A & (\omega_{AB} < 0, \tau_A \geq 0) \\ (1-1/0.97)\tau_A & (\omega_{AB} < 0, \tau_A < 0) \end{cases} \quad (6)$$

where, τ_A is the sun gear torque and ω_{AB} is the difference in the speed of the sun gear and the carrier of the planetary gear.

Figure 4 shows a simulation result to investigate the torque distribution ability of the TVD. It became clear that this TVD unit has capability of distributing the driving torque between right and left wheels according to the input torque of the control motor and the torque distribution ratio can be bigger than ordinary LSD (limited slip differential) gear set if the mechanical strength is enough to cope with the maximum torque. The torque distribution ability is thus only limited by mechanical strength of the gear sets and the ability of the control motor.

Figure 5 shows an example of calculation result of each gear speed of the TVD. It was confirmed that this result coincide with the actual motion.

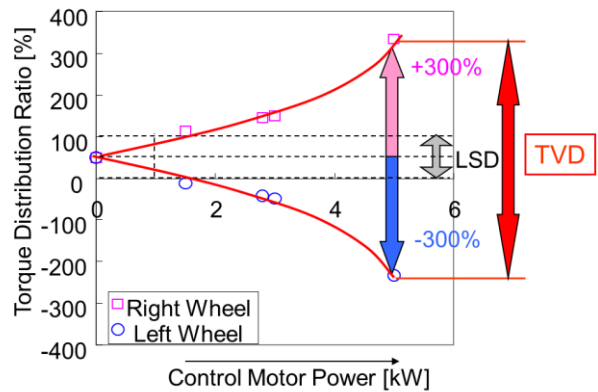


Figure 4. Torque distribution ability of TVD

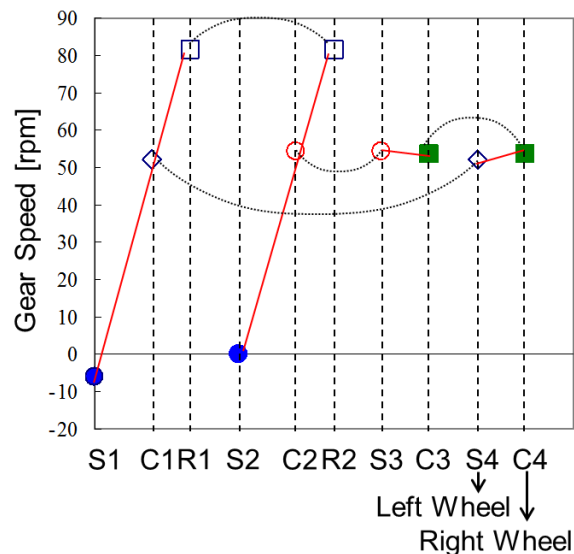


Figure 5. Example of TVD gear speed calculation

3.4 Electrical Model of Motor and Inverter

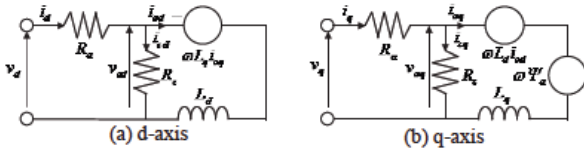


Figure 6. Equivalent circuits of each motor

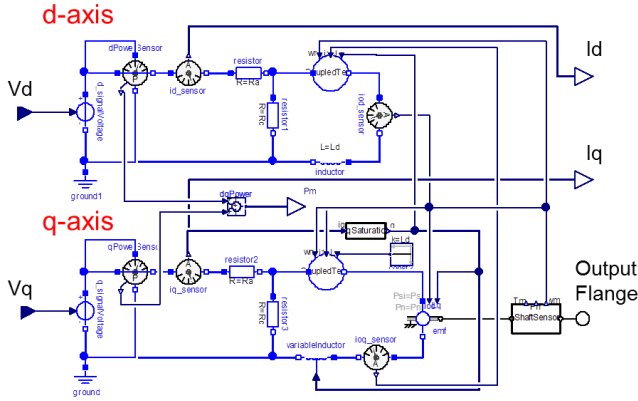


Figure 7. Electrical motor model by Modelica

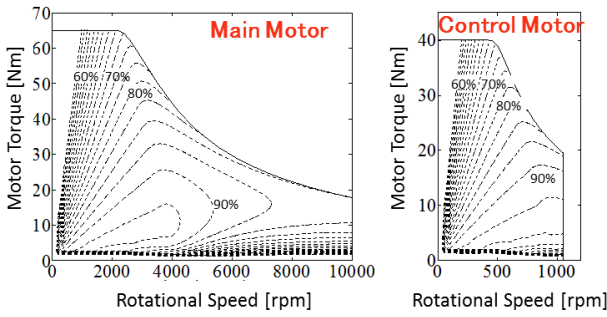


Figure 8. Motor characteristics

Both the main and control motors installed in the target EV are permanent magnet synchronous motors. The equivalent circuits of these motors can be expressed as shown in Figure 6 as d-axis and q-axis direct current (DC) circuits (Park, 1933). It is not necessary to describe the explicit equations here since modeling can be performed simply by laying out each device as shown in Figure 7 using freely available electric circuit library of Modelica Standard Libraries (MSL). Concurrently copper loss L_{Cu} and iron loss L_{Fe} of each motor are calculated using following equations (Inoue et al., 2014).

$$L_{Cu} = R_a I_a^2 = R_a (i_d^2 + i_q^2) \quad (7)$$

$$L_{Fe} = \frac{v_{od}^2 + v_{oq}^2}{R_c} = \frac{\omega_e^2 [(L_d i_{od} + \Psi_a)^2 + (L_q i_{oq})^2]}{R_c} \quad (8)$$

$$i_d = i_{od} + i_{cd}, \quad i_q = i_{oq} + i_{cq} \quad (9)$$

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = R_a \begin{bmatrix} i_{od} \\ i_{oq} \end{bmatrix} + \left(1 + \frac{R_a}{R_c}\right) \begin{bmatrix} v_{od} \\ v_{oq} \end{bmatrix} + p \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \begin{bmatrix} i_{od} \\ i_{oq} \end{bmatrix} \quad (10)$$

$$\begin{bmatrix} v_{od} \\ v_{oq} \end{bmatrix} = \begin{bmatrix} 0 & -\omega_e L_q \\ \omega_e L_d & 0 \end{bmatrix} \begin{bmatrix} i_{od} \\ i_{oq} \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_e \Psi_a \end{bmatrix} \quad (11)$$

where, v_d and v_q are the voltage of the d and q axis, respectively, i_d and i_q are the current of the d and q axis, respectively, ω_e is the electric angular velocity, R_a is the winding unbalance voltage attenuation, R_c is the equivalent iron loss resistance, L_d and L_q are the inductance of the d and q axis, respectively, and Ψ_a is the inter-linkage magnetic flux. Also the motor characteristics of efficiency according to motor torque and rotational speed are considered as shown in Figure 8.

The inverter can be handled simply as a component that generates loss L_{inv} proportionally to the current vectors of the motors as follows.

$$L_{inv} = \kappa I_a \quad (12)$$

3.5 Mechanical Model of Suspension and Body

3D multi-body dynamic system (MBS) models of suspension, steering and body were installed to calculate vehicle dynamics characteristics. Suspension model was constructed as an assembled model of each suspension linkage, joints and force elements such as spring, damper and bushing. Non-linear tire model based on 'Magic Formula' model (Pacejka02) was used to calculate combined lateral force and longitudinal force of each tire. Steering model considered the characteristics of viscous friction of steering gear box and steering shaft as well as steering shaft stiffness. By these detailed models, it became possible to analyze the effects of steering angle change and camber angle change caused by vehicle roll, side force and tire aligning torque.

Figure 9 shows a comparison of simulation results and experimental test results about camber angle change by wheel bump displacement and steering angle change by tire aligning moment. It was confirmed that the simulation results matched with the experimental results with good consistency.

Figure 10 shows an analysis result about the effect of suspension characteristics to cornering compliance coefficient normalized by the effect of tire slip angle change for one example of a front double wish-born suspension. It became clear that the effect of the tire aligning torque to tire toeing angle is relatively large than other design indexes.

Also 6 degree-of-freedom motion of the vehicle body was calculated by considering all the reaction forces and torques acting at suspension upper support and all of the connection portions of the linkages. Additionally 3D MBS model of TVD gear unit mounts was applied in the vehicle dynamics model. And rotational stiffness of the drive shafts was also considered. This feature enabled calculation of the body motion (mainly pitching motion) caused by the

reaction of driving torque and oscillation caused by resonance of tire rotational stiffness, drive shaft stiffness and differential gear mount stiffness.

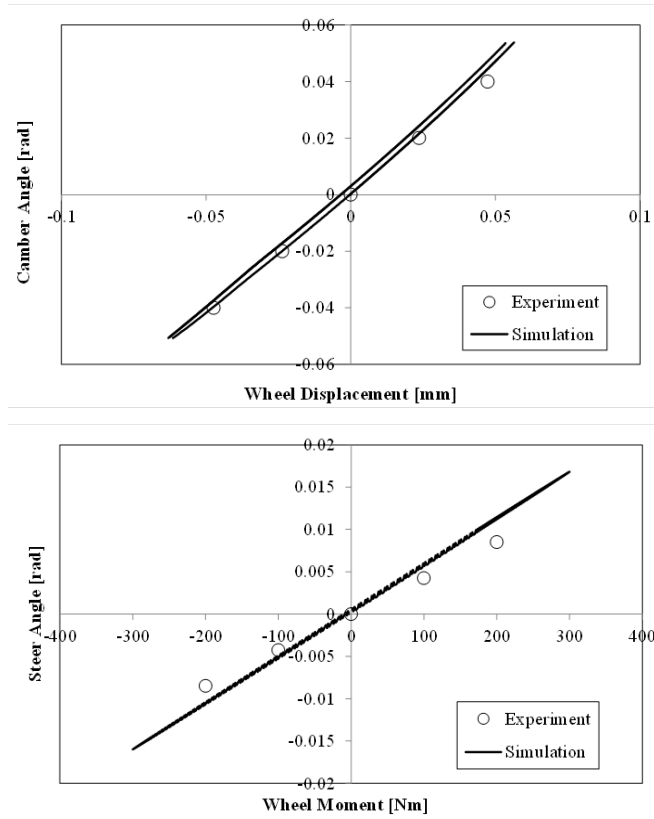


Figure 9. Example of suspension characteristics

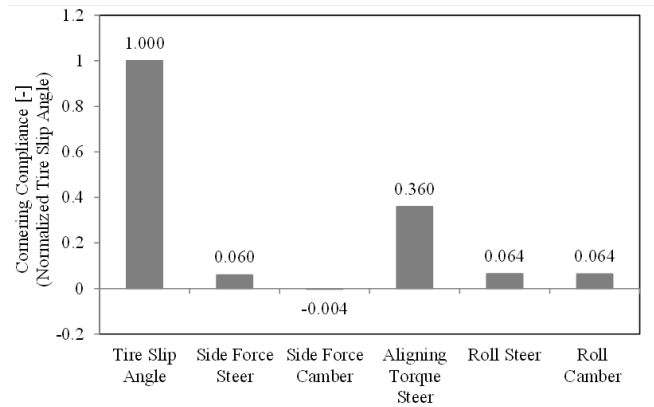


Figure 10. Effect of suspension characteristics to cornering compliance coefficient. (Normalized by the effect of tire slip angle.)

3.6 Model of DYC Controller

The control element generates the torque command values of the main motor and control motor. These command values are then input into the motor models. Two kinds of DYC controller were researched. Yaw rate feedback control to let the vehicle yaw rate follow the desired yaw rate for stabilizing crosswind disturbances comprises the control laws shown in

Figure 11. Slip angle feedback controller shown in Figure 12 aims to let the vehicle slip angle to zero to stabilize the vehicle attitude while cornering and lane change.

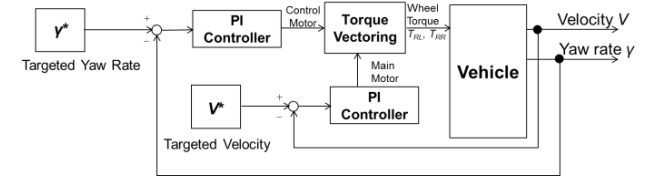


Figure 11. Yaw rate feedback controller of DYC

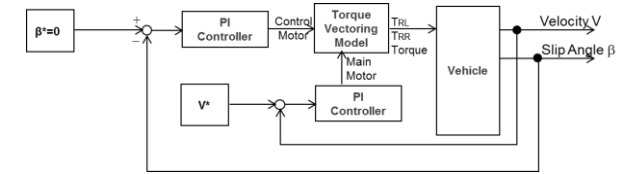


Figure 12. Slip angle feedback controller of DYC

In both controllers, the main motor performs feedback control by proportional and integral (PI) control of the vehicle speed because the vehicle speed is dominant for the total driving torque supplied by the main motor. The control motor performs feedback control by PI control of the yaw rate and vehicle slip angle respectively. As shown in Figure 4, control motor changes the distribution of left wheel torque T_{RL} and right wheel torque T_{RR} . Therefore the vehicle motion can be changed by yaw moment generated by the torque difference between left wheel and right wheel. In general, the vehicle motion can be estimated by single track model of vehicle dynamics described by equation (13) and (14).

$$\frac{d}{dt} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} -\frac{c_f + c_r}{MV} & -1 - \frac{a_f c_f - a_r c_r}{MV^2} \\ -\frac{a_f c_f - a_r c_r}{I_z} & -\frac{a_f^2 c_f + a_r^2 c_r}{I_z V} \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} \quad (13)$$

$$+ \begin{bmatrix} \frac{c_f}{MV} & \frac{c_r}{MV} \\ \frac{a_f c_f}{I_z} & -\frac{a_r c_r}{I_z} \end{bmatrix} \begin{bmatrix} \delta_f \\ \delta_r \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{I_z} \end{bmatrix} N$$

$$N = w(T_{RL} - T_{RR}) / r_t \quad (14)$$

Here,

- β : Vehicle slip angle [rad]
- γ : Yaw rate [rad/s]
- M : Vehicle mass [kg]
- V : Vehicle speed [m/s]
- I_z : Vehicle yaw moment of inertia [kgm²]
- a_f : Length between front axle and CG [m]
- a_r : Length between rear axle and CG [m]
- l : Wheel base (= $a_f + a_r$) [m]
- c_f : Front tire cornering power [N/rad]
- c_r : Rear tire cornering power [N/rad]
- δ_f : Front tire steering angle [rad]

- δ_r : Rear tire steering angle [rad]
 N : Direct yaw moment [Nm]
 r_t : Tire radius [m]
 w : Tread [m]

3.7 Model of Energy Consumption by Electric Drives

Energy consumption in mechanical part (= gear train), electrical part (= motor, inverter) are calculated by using following equations.

$$P_e = P_m + L_{mj} + \sum_{j=1}^2 L_{ej} \quad (15)$$

$$P_m = P_v + I_z \gamma = P_{rr} + P_{ar} + P_{sy} + P_{sx} + I_z \gamma \quad (16)$$

$$L_{mj} = 0.95 \times (T_M + T_C) \quad (17)$$

$$L_{ej} = L_{Cuj} + L_{Fej} + L_{Invj} \quad (18)$$

Here, P_e is the sum of the energy consumption, P_m is the total mechanical work using driving resistance power defined by equation (1), L_{mj} is the TVD mechanical loss, and L_{ej} is the electrical loss of motor and inverter. L_{Cuj} is copper loss, L_{Fej} is iron loss and L_{Invj} is switching loss of inverter respectively. Here, $j=1$ refers to the main motor and $j=2$ refers to the control motor. Since it is difficult to accurately calculate the TVD mechanical loss including all kinds of friction, the calculation assumes a constant overall efficiency of 95% of main motor torque T_M and control motor torque T_C .

Figure 13 shows the calculation results using the full-vehicle model in steady-state cornering with a turning radius of 60 m and a vehicle speed of 40 km/h. The calculated results using the full-vehicle model closely matched the theoretical results calculated based on Equations (15) and (16), thereby confirming the validity of this model. It is shown that electrical loss increases much when large DYC torque is applied.

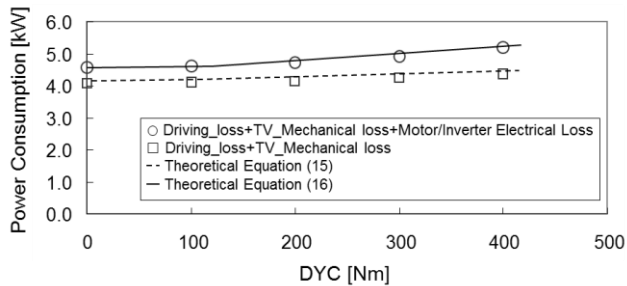


Figure 13. Comparison of energy consumption

4 Simulation Results

4.1 Steady State Cornering

The simulation of steady state cornering assumed running condition of a turning radius R of 135 m and a vehicle speed V of 60 km/h. Figure 14 shows the

relationship between the vehicle slip angle and energy consumption in the $\beta=0$ control. Figure 15 shows the same relationship for the yaw rate feedback control. Here, the target yaw rate was calculated by following equation.

$$\gamma^* = K_{yaw_gain} \times \frac{V}{R} \quad (19)$$

In the case of both controls, the turning resistance became lower as the slip angle and steering angle decreased. As a result, the power of the main motor decreased (point “i” in Figure 14 and Figure 15). In contrast, application of TVD generated mechanical loss, which resulted in an overall increase in energy consumption due to the energy consumption of the control motor (point “ii” in Figure 14 and Figure 15). Furthermore, the TVD mechanical loss was lower than the electrical loss. This result indicates that, in this configuration, a reduction in motor/inverter electrical loss is extremely important for reducing energy consumption. Finally, further examination of the $\beta=0$ control in Figure 14 shows that a very slight vehicle slip angle remains when the control is applied (point “iii” in Figure 14). Focusing on the TVD control motor torque shows that, in an ideal condition without TVD or motor/inverter loss (Figure 16), the vehicle slip angle is zero because the power of the control motor does not exceed the maximum possible output torque of 40 Nm. However, after factoring in each type of loss, the control motor power becomes saturated (point “iv” in Figure 16). In this way, integrating different physical models into a single model enables quantitative studies of the effects of each type of loss on vehicle dynamics and control.

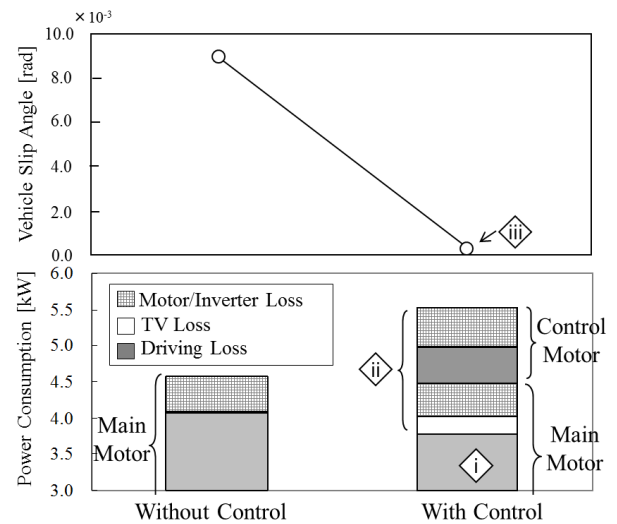


Figure 14. Energy consumption with $\beta=0$ control

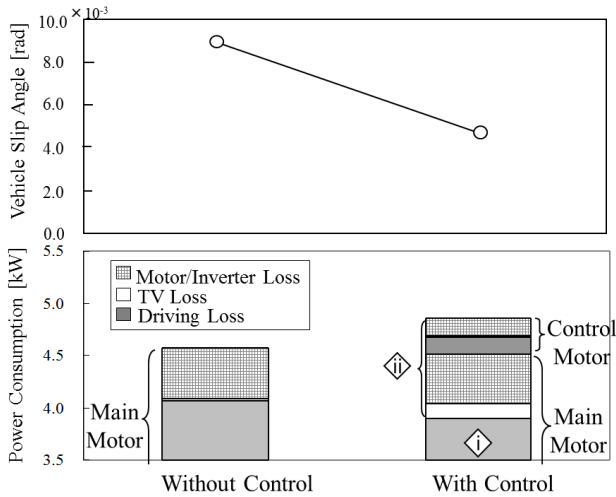


Figure 15. Energy consumption with yaw rate feedback Control

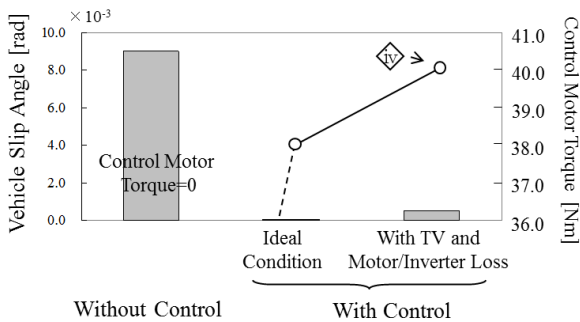


Figure 16. Relationship between vehicle slip angle and control motor torque with $\beta=0$ control

4.2 Winding Road Driving

Actual driving conditions generally feature many turns and the proposed EV is also likely to be driven while utilizing controls to actively enhance dynamic performance. Therefore, to simply evaluate performance under real-world driving conditions, a study was performed on winding roads to simulate actual continuous steady-state cornering. The study simulated an actual 8.1 km winding road course, which combines straight sections and gradients.

First, the course was constructed using the commercially available CarMaker software. Driving behavior was predicted using a driver model of CarMaker assuming a constant speed of 60 km/h. Developed Modelica model of TVD was connected with CarMaker using FMI (Functional Mockup Interface). Only vehicle speed was set in the driver model and lateral acceleration was calculated during the simulation. By using CarMaker as a virtual driving test platform, it became possible to combine the good realistic driver model of CarMaker and the detailed drive train model made by using Modelica.

Finally, the energy consumption and steering wheel angle by the driver model were predicted using the time-series data for lateral acceleration in Figure 17.

Figure 18 shows the time-series data for the total energy consumption and steering wheel angle with a yaw gain ratio of 1.5. (Abbreviation of ‘W/O Control’ means ‘Without Control’.) Although yaw gain control causes an increase in total energy consumption, the steering wheel angle decreases. These prediction results facilitate the identification of the optimum control gain with respect to a set system configuration, assuming real-world driving conditions.

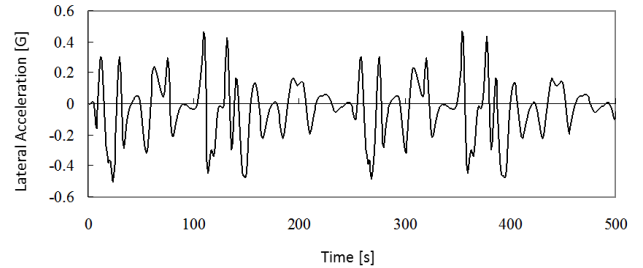


Figure 17. Time-series data for lateral acceleration

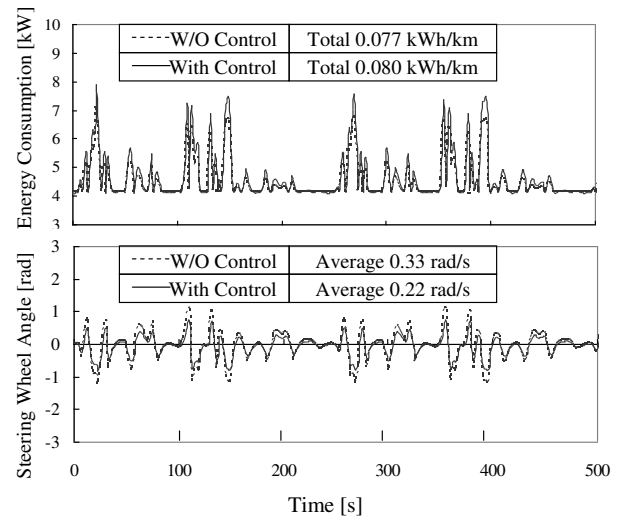


Figure 18. Time-series data (upper: energy consumption, lower: steering wheel angle)

5 Conclusions

This paper described the development of a full-vehicle model to quantitatively evaluate the relationship between vehicle dynamics with DYC input and energy consumption for a small EV. The following conclusions were obtained.

- (i) TVD drivelines with several planetary gear sets and motor/inverters with multiple electrical elements can be constructed simply in one model using Modelica.
- (ii) The model was able to quantitatively identify the breakdown of energy consumption increases and decreases for achieving the target vehicle dynamics. In addition, it was found that reducing motor loss makes a larger

contribution to lower energy consumption than reducing TVD mechanical loss.

- (iii) The real-world energy consumption and driver's workload (steering wheel angle) through DYC was predicted on a simulated course based on actual winding roads.

For future works, it is planned to consider the effect of drive shaft stiffness for TVD control. Also controlling tire slip by motor torque as well as maximizing regeneration by braking is essential to expand the capability of electric control of optimal tire slip control. This work will also be useful to design a proper system configuration of mechanical break and electric break and also designing proper torque blending control.

References

- DLR, PowerTrain Library Users Guide (Version 2.1.0), 2013
- Y. Hirano, S. Inoue and J. Ota, Model-based Development of Future Small EVs using Modelica, *Proceedings of Modelica Conference 2014*, 2014.
- B. Höhn et al., Torque Vectoring Driveline for Electric Vehicle, *Proceedings of the FISITA 2012 World Automotive Congress*, Vol. 191, pp. 585-593, 2013.
- S. Inoue, J. Ota, Y. Hirano, T. Kobayashi, A. Kawaguchi and H. Sugiura, Study on Full-Vehicle Model Integrating Vehicle Dynamics and Energy Consumption, *Proceeding of 12th International Symposium on Advanced Vehicle Control (AVEC'14)*, 20149329, 2014.
- T. Kobayashi, E. Katsuyama, G. Sugiura, E. Ono, M. Yamamoto, A research about driving force distribution control and energy consumption while cornering, *Proceeding of 2013 JSAE Annual Congress (Spring)*, 352-20135393, 2013 (in Japanese).
- Modelon, A.B., Vehicle Dynamics library Users Guide (Version 1.8), 2014.
- R.H.Park, Two-reaction Theory of Synchronous Machines: Part II, *AIEE Trans.*, Vol.52, pp.352-355, 1933.
- C. Pelchen et al., Modeling and Simulating the Efficiency of Gearboxes and of Planetary Gearboxes, *Proceedings of 2nd International Modelica Conference*, pp. 257-266, 2002.

Modeling of Torque Vectoring Drives for Electric Vehicles: a Case Study

Franciscus L.J. van der Linden Jakub Tobolář

German Aerospace Center (DLR), Institute of System Dynamics and Control, 82234 Wessling, Germany
{Franciscus.vanderLinden, Jakub.Tobolar}@dlr.de

Abstract

This paper shows some aspects of the implementation of a gear model with losses, nonlinear elasticity and forcing errors in the Modelica language utilizing concept of replaceable functions. Using such gear model for a torque vectoring drive modeling, a case study about a powertrain dynamic behavior in a simplified vehicle model is carried out. The total vehicle model is analyzed in several detail stages of the powertrain reaching from a fixed efficiency with constant spring stiffness to a model using nonlinear losses and nonlinear tooth stiffness. Subsequently, the simulation results of such levels of modeling detail proving tendency to drive line oscillation are presented and discussed.

Torque Vectoring Drive, Gearing, Vehicle Dynamics

1 Introduction

To design the gearing solution for an electrical vehicle, different gear topographies are typically analyzed utilizing computer simulations in an early design stage. To perform such studies efficiently, it is important that the gearing topographies can easily be designed and integrated into the vehicle models which are used for maneuverability tests assessing the driving quality.

A solution enabling such gear topography design and vehicle integration was introduced recently by (van der Linden, 2015). To prove the usability of that concept also for more complex gearing topographies, an electric vehicle powertrain configuration with controlled torque vectoring device was chosen in this paper – a future-oriented solution particularly suitable to actively influence the dynamic behavior of the vehicle, such as using active yaw rate control. Such a torque vectoring drive (TVD) configuration is used e.g. in experimental cars like the VISIO.M (Gwinner et al., 2014) and allows for very high vectoring torques with a small electric motor.

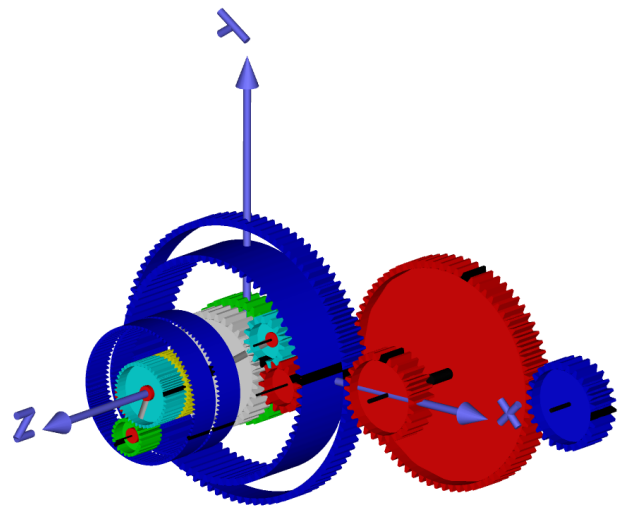


Figure 1. Torque vectoring drive consisting of a differential, superimposing unit and spur gear train. Note that only single planets are shown for simplification of the calculations.

A graphical overview of the gearing solution is shown in Figure 1.

After giving an overview of some implementation aspects of the method in Section 2, the present study will continue with a TVD model description in Section 3. Here, a gear with constant elasticity and constant efficiency will be first introduced as a reference model. For further investigations, the model complexity will be successively increased with different loss models as well as nonlinear elasticity and backlash in the gearing. Utilizing a simple vehicle model, briefly referred in Section 4, the simulation results will be discussed in detail in Section 5.

2 Gear model description

The gear models used in this analysis base on previous work which consisted of the simulation of elastic ideal gears (van der Linden, 2012). These models have been extended with various loss models and elasticity models according to (van der Linden,

2015). An overview of the forces and torques calculated in this publication are shown in Figure 2. The forces and torques (F_{xA} , F_{yA} , τ_A , F_{xB} , F_{yB} and τ_B) in this Figure are calculated using the integral of the forces of a complete cycle of the meshing tooth. By supplying also an internal gear force element, epicyclic gears can be modeled as well. Since the derivation of all the theory goes beyond the scope of this paper, only brief outline is given in the following sections. For a detailed description, please refer to the abovementioned paper. Since the derived model of gear teeth contact is purely planar, it is implemented using the PlanarMechanics toolbox (Zimmer, 2012). This allows the use of standard planar parts and enables a good transferability of forces and torques into the 3D world

2.1 Friction force implementation

Since in gear dynamics different friction models are often used, a decision was made to implement the friction using a replaceable function structure.

The friction is implemented using a state machine to be able to handle friction during the **Forward**, **Backward** and **Stuck** mode. To switch between these modes, two transition modes are used: **StartForw** and **StartBackw**. A tearing variable **sa** is used in the Stuck mode to calculate the forces to keep the model stuck. This approach is similar to the friction implementation of (Otter et al., 1999). The gear friction force F_t is calculated using specialized functions based on e. g. gear meshing speed, operational mode of the gear contact, contact angle and the radii of the gear wheels.

The concept of replaceable functions allows for a quick selection between the different friction models: no friction, viscous friction, specified efficiency, Coulomb friction, friction according to the DIN 3990 specifications (DIN 3990 Teil 4, 1987) and a friction implementation from Niemann and Winter (1989). The DIN 3990 friction and the friction to Niemann and Winter both define the friction as a function of the speed and loading of the gear.

Furthermore, a continuous friction model which is not based on a state machine is implemented. This implementation uses a regularized friction model to smooth the discontinuity of the gear friction. It is implemented using

$$F_t = \mu |F_n| \tanh \frac{|v_{mesh}|}{v_{mesh,0}}. \quad (1)$$

In this equation, v_{mesh} is the relative speed of the gears at the meshing point and $v_{mesh,0}$ the characteristic meshing speed which is chosen small compared to the nominal meshing speed. Using this regularization, event chattering of gear systems with many gear contacts can be avoided. However, it must be

noted that in this case no stiction can take place, as the friction is zero at zero speed.

In this paper, a fixed friction coefficient will be used as this method is heavily used in the design of gear transmissions for powertrain analysis, together with the friction implementation to the DIN 3990 norm due to a good match with measured friction results in a previous publication (van der Linden, 2015).

2.2 Elasticity implementation

Similar to the variability of friction methods used in the modeling of gears, also the gear elasticity is described in many ways. In most cases, a nonlinear relation between normal forces F_n and deformation of the gear at contact is present. To incorporate the different stiffness models known from literature, also the elasticity is implemented using another set of replaceable functions. These functions calculate the normal contact force F_n from multiple model inputs like the mesh deformation and speed, gear radii, thickness of the wheels and wheel positions. The position of the gear wheels makes it possible to include a position dependent gear stiffness which can be used to simulate the effect of meshing teeth or the effect of a damaged tooth.

2.3 Forcing error implementation

To simulate forcing errors like misalignment of the gear wheels, manufacturing errors or damaged tooth, a position dependent forcing error is added to the overall gear deformation. In Figure 3, the deformation between the gears is given by $\Delta_{AB} = \Delta_{AB,0} + \Delta_{AB,e}$. In this equation, $\Delta_{AB,e}$ is the elastic deformation of the gear contact as discussed in Section 2.2. Adding the forcing distance $\Delta_{AB,0}$ gives the total gear deformation.

Also in this case, replaceable functions are used to implement several cases: a forcing error defined by the misalignments of the gears, a forcing error defined by a Fourier-series and a table-based interpolation. All these methods define the forcing error as a function of the position of each gear wheel.

2.4 Graphical representation of gears

The graphical representation of the gears is an important way to check proper geometry of the gear. Therefore, visualizers from `Modelica.Mechanics.MultiBody.Visualizers` are used to visualize the gear wheels. The results of such exemplary 3D representations can be seen in Figure 1 and Figure 5. The parameters needed for the visualization, such as gear radius or thickness, are directly taken from the gear model parameters.

Table 1. TVD model configurations under investigation

Configuration	Description
Fixed eta	Constant spring constant, constant gear efficiency
Fixed eta with Backlash	Nonlinear spring constant and backlash, constant gear efficiency
DIN 3990 eta with Backlash	Nonlinear spring constant and backlash, efficiency using to DIN 3990

TVD is generated as shown in Figure 8. To represent the connection elasticity between the drives, rotational stiffness-damping elements are added from the Modelica standard library.

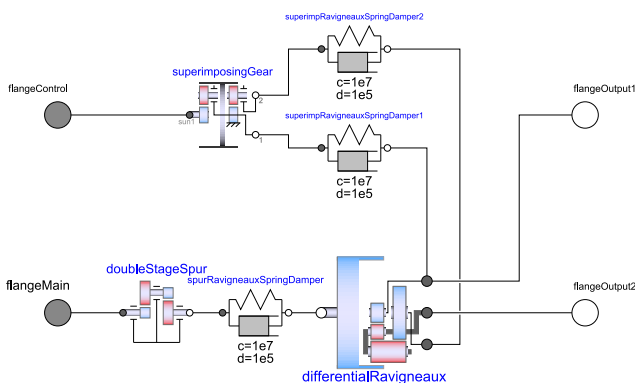
In the overall TVD model, ten gear connections are included.

In the simulations which are presented in Section 5, three different configurations are analyzed according to Table 1.

The spring constant of the gear instances is set to 20×10^6 N/m/mm (Newton per meter per millimeter gear width), and the gear damping to 20×10^3 Ns/m/mm for all gear connections. The coupling between the superimposing gear, Ravigneaux gear and spur gear train have a stiffness of 10^7 Nm/rad and a damping of 10^5 Nms/rad, respectively.

4 Vehicle model

To analyze the TVD model in typical vehicle driving maneuvers, a vehicle model has to be utilized. For the sake of simplicity, a planar vehicle model was introduced which moves in the horizontal plane, thus enabling longitudinal, lateral and yaw motion only. Additionally, a six degrees of freedom mass (i. e. three positions and three rotations) was joined to the planar vehicle body. By taking into account


Figure 8. Complete TVD consisting of Ravigneaux differential, superimposing gear and spur gear train

the forces on this mass, wheel load variation due to vehicle mass transformation between wheels during braking, accelerating and cornering are enabled. The tire models allow slip and are based on the work of Zimmer and Otter (2010).

A small size electric vehicle with rear-wheel drive is considered for simulation. Its mass is about 1000 kg with wheelbase of 2.6 m and track of 1.45 m.

The powertrain of the vehicle consists of TVD as described above, the main motor which applies the main driving torque, and a differential motor which divides torques to the wheels of one axle. Utilizing the differential motor control, the torque vectoring functionality can be realized. Finally, driveshaft elements are considered as well to additionally incorporate their elasticity. An overview of the model is shown in Figure 9. To mimic the electrical time constant of the motors, a first order system with a time constant of 10^{-3} s for both motors is used.

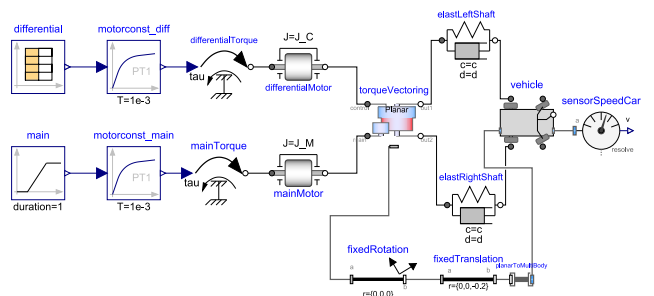
5 Simulation results

Using the vehicle model with a free steering setup (free steering wheel), an acceleration maneuver is simulated. The main motor torque is given as a ramped signal, and the differential motor torque as a changing signal as shown in Figure 10.

5.1 Elastic drive shafts

The wheel torque of the right driveshaft during the maneuver is shown in Figure 11. It can be observed that the results of the constant efficiency and the DIN 3990 efficiency are differs significantly. The fixed efficiency cases (97% per gear stage) show a behavior which is intuitively expected of the TVD: the differential torque of the differential motor is amplified and split between the two axles.

Introducing the DIN 3990 friction model, the results yield – in contrast to the constant efficiency – an oscillating output torque. This is caused by the fact that due to the pre-load of the differential,


Figure 9. Vehicle model with motor configuration and driveshaft elasticity. The right bottom section of the diagram (with the planarToMultibody element) enables an animation where the drive is fixed to the car.

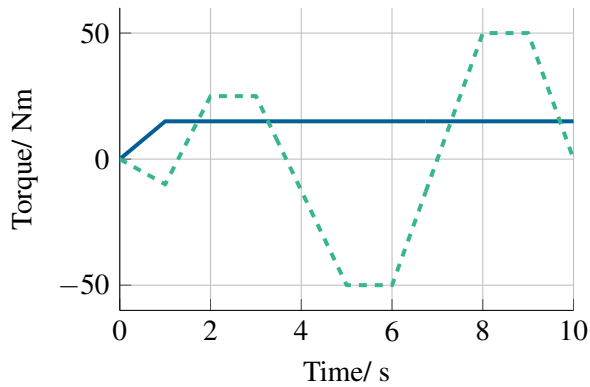


Figure 10. Motor torques during maneuver. The main motor torque (—) has a ramp of 1 s to 15 N m, the differential motor (---) is controlled with a changing reference torque.

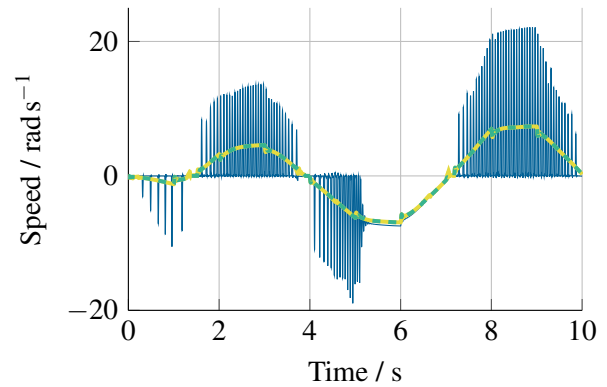


Figure 12. Speed of the differential motor using driveshafts with the reference stiffness. Different friction and elasticity models are shown: A fixed efficiency without backlash (---), fixed efficiency with backlash (—) and a friction law to the DIN 3990 standard (—).

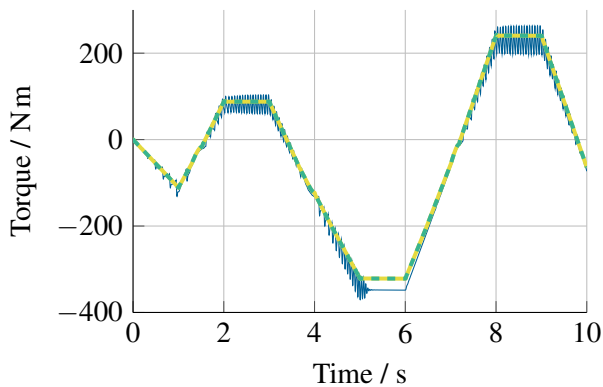


Figure 11. Wheel torques of the right driveshaft (reference stiffness) with different friction and elasticity models: A fixed efficiency without backlash (---), fixed efficiency with backlash (—) and a friction law to the DIN 3990 standard (—).

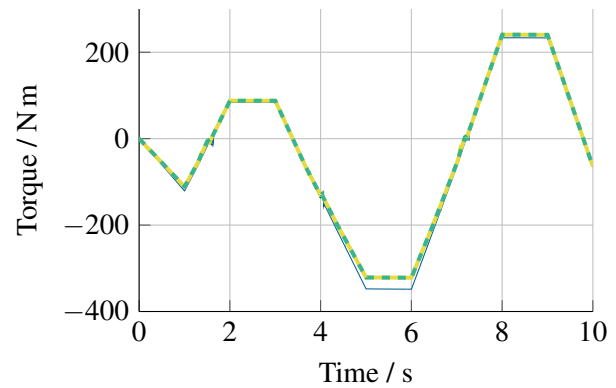


Figure 13. Wheel torques of the right driveshaft with different friction and elasticity models. A ten times increased stiffness of the driveshafts is used. A fixed efficiency without backlash (---), fixed efficiency with backlash (—) and a friction law to the DIN 3990 standard (—).

combined with low rotational velocities, lead to high friction. Due to this high friction combined with the pre-load, the gear can get in the stuck mode (this phenomenon is also described and measured for air path actuators by Ahmed et al. (2012)).

The combination of the drivetrain elasticity with high friction leads to a stick-slip problem resulting in highly varying torques. Note that without further measurements and / or experience on TVD, it cannot be concluded which of the friction models correctly represents the real system.

Analyzing the speed of the differential motor depicted in Figure 12, the stick-slip problem is also evident. High rotational accelerations are caused by the fast variation of the motor speed, which can lead to high loads on the rotor of the motor. This can lead to fatigue damage of the motor.

5.2 Stiff driveshafts

Using driveshafts with a significant higher stiffness and damping, the stick-slip problems described in Section 5.1 can be avoided. In presented example with increased stiffness, a ten times higher friction and damping has been used w.r.t. the nominal situation. The wheel torques of the right rear wheel (see Figure 13) behave as expected, also for TVD using the DIN 3990 friction model. Moreover, the high peaks in motor velocity of the differential gear are eliminated, cf. Figure 12 and Figure 14.

5.3 Simulation of eccentricities

Eccentricities are common in most gear wheels and are often caused by manufacturing tolerances. To simulate a non-perfect drive, an eccentricity of 10 μm is added to both gear wheels of the first stage of the spur gear train. This eccentricity excites the

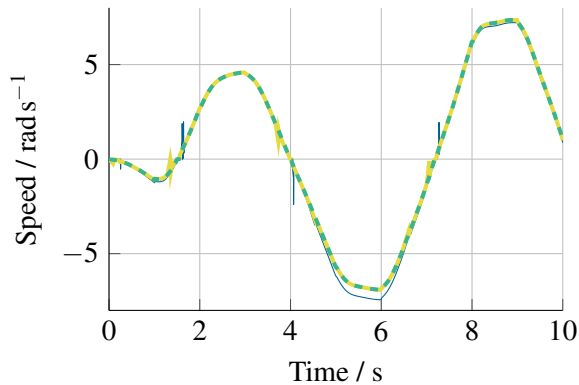


Figure 14. Speed of the differential motor using driveshafts with ten times increased stiffness. Different friction and elasticity models are shown: a fixed efficiency without backlash (---), fixed efficiency with backlash (—) and a friction law to the DIN 3990 standard (—).

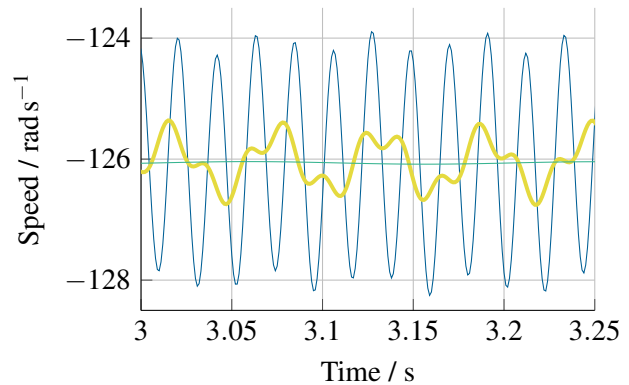


Figure 16. Speed of the differential motor with and without gear eccentricities: All simulation results have a fixed efficiency and a nonlinear stiffness. The different lines show a simulation without eccentricity(—), simulation with the nominal stiffness (—) and a simulation with an increased stiffness(—).

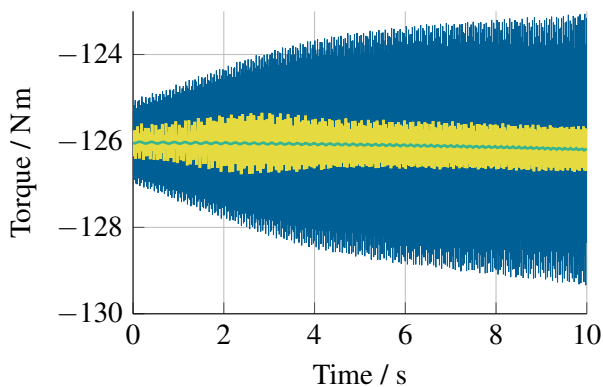


Figure 15. Wheel torques with and without gear eccentricities: All simulation results have a fixed efficiency and a nonlinear stiffness. The different lines show a simulation without eccentricity(—), simulation with the nominal stiffness (—) and a simulation with an increased stiffness(—).

gear train leading to vibrations. In this simulation, a constant torque of 5 Nm is applied to the differential motor to keep the Ravigneaux differential out of the stuck mode. The main drive motor is driven with a constant load for a constant acceleration.

In Figure 15, the wheel torques of a simulation with a stiff driveshaft with eccentricity, an elastic driveshaft with eccentricity and an elastic driveshaft without eccentricity are given. Analyzing the simulation results, it is clear that this eccentricity has a high-frequent impact on the wheel torques. With a nominal driveline, the torque variations are lower at high velocities as for a stiff driveline. A detailed view of this vibration is shown in Figure 16. The high frequent vibrations of the gear seem to excite the drivetrain for the nominal stiffness drivetrain.

6 Discussion

During performed simulations, we realized that – due to the large number of switching components and high gear stiffness – the proposed model challenges common numerical solvers like DASSL or Radau IIA. Finding consistent restart conditions after an event can be hard, since this often directly triggers a next event in an adjacent gear connection.

In some cases, it is therefore advisable to use a regularized friction model as presented in Section 2.1. Such friction models can help to avoid events and make a simulation progress even if very complex gearing configurations are analyzed. However, most of these problems can be avoided by modeling the real-life world more accurately. As an example, in this paper, spring-damper models between the differential, superimposing gear and spur gear train have been added to mimic the stiffness of the connections. This avoided many problems with the simulation results.

7 Conclusion

In this paper, different techniques for gear modeling were presented and adopted for a torque vectoring drive which was analyzed in complete car model simulations. Applying such gear modeling techniques, which include losses, nonlinear elasticity and forcing errors, a various level of gear detail can be selected which proved to significant influence the simulation results.

The higher level of modeling detail is particularly important when investigating torque and speed oscillation issues which can be useful for e. g. driveline design. Then, simple fixed efficiency based friction models are insufficient. In contrast, DIN 3990 or

similar friction models are required to capture such effects. Furthermore, it was shown that both insufficient torsional stiffness of the drive shafts and stick slip in the gear lead to such large torque oscillations within a complete driveline.

The influence of gear eccentricities on the driveline was shown for driveshafts of different elasticity. It is shown that the a higher stiffness of this shaft increases the load on this axle. This shows that a trade-off must be made between the load caused by eccentricities and the load caused by the stick-slip effect, as a high driveshaft elasticity lowers the load caused by stick-slip effects, but increases the load caused by a stiffer shaft.

The drawback of some presented models can be an increased simulation effort due to large number of events. For such cases, the regularized friction model proved to be possible alternative.

It is worth mentioned that the model and the simulation results were not validated so far. Especially, the damping coefficient is largely unknown and not well researched at the moment. It is advisable to push the experimental research to get a usable estimate of the gear damping properties in the future.

References

- F. S. Ahmed, S. Laghrouche, and M. El Bagdouri. Overview of the modelling techniques of actuator nonlinearities in the engine air path. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 227(3):443–454, September 2012. ISSN 0954-4070. doi:10.1177/0954407012453905.
- DIN 3990 Teil 4. Tragfähigkeitsberechnung von Stirnrädern; Berechnung der Freßtragfähigkeit, 1987.
- Philipp Gwinner, Michael Otto, and Karsten Stahl. Lightweight Torque-Vectoring Transmission for the Electric Vehicle VISIO.M. In *COFAT 2014*, March 2014. URL <http://mediatum.ub.tum.de/doc/1226683/1226683.pdf>.
- Gustav Niemann and Hans Winter. *Maschinenelemente: Band 2: Getriebe allgemein, Zahnradgetriebe - Grundlagen, Stirnradgetriebe (German Edition)*. Springer, 1989. ISBN 3-540-11149-2.
- M Otter, H Elmqvist, and S E Mattsson. Hybrid modeling in Modelica based on the synchronous data flow principle. In *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pages 151–157, 1999. doi:10.1109/CACSD.1999.808640.
- Franciscus L. J. van der Linden. Modelling of Elastic Gearboxes Using a Generalized Gear Contact Model. In *Proceedings of the 9th International MODELICA Conference*, pages 303–310, Munich, November 2012. Linköping University Electronic Press. doi:10.3384/ecp12076303.
- Franciscus L. J. van der Linden. Modeling of geared positioning systems: An object-oriented gear contact model with validation. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, June 2015. ISSN 0954-4062. doi:10.1177/0954406215592056.
- Dirk Zimmer. A Planar Mechanical Library for Teaching Modelica. In *Proceedings of the 9th International Modelica Conference*, pages 681–690, Munich, November 2012. Linköping University Electronic Press. doi:10.3384/ecp12076681.
- Dirk Zimmer and Martin Otter. Real-time models for wheels and tyres in an object-oriented modelling framework. *Vehicle System Dynamics*, 48(2):189–216, February 2010. ISSN 0042-3114. doi:10.1080/00423110802687596. URL <http://www.tandfonline.com/doi/abs/10.1080/00423110802687596>.

Co-Simulation of Hybrid Systems with SpaceX and Uppaal

Sergiy Bogomolov¹ Marius Greitschus² Peter G. Jensen³ Kim G. Larsen³
Marius Mikučionis³ Thomas Strump² Stavros Tripakis⁴

¹IST Austria, Austria

²University of Freiburg, Germany

³Aalborg University, Denmark

⁴Aalto University, Finland, and University of California, Berkeley, USA

Abstract

The Functional Mock-up Interface (FMI) is an industry standard which enables co-simulation of complex heterogeneous systems using multiple simulation engines. In this paper, we show how to use FMI in order to co-simulate hybrid systems modeled in the model checkers SPACEEX and UPPAAL. We show how FMI components can be automatically generated from SPACEEX and UPPAAL models. We also validate the co-simulation approach by comparing the simulations of a room heating benchmark in two cases: first, when a single model is simulated in SPACEEX; and second, when the model is split in two submodels, and co-simulated using SPACEEX and UPPAAL. Finally, we perform a measurement experiment on a composite model to show a potential for statistical model checking using stochastic co-simulations.

Keywords: FMI, hybrid system, timed automaton

1 Introduction

Despite advances in model checking and other formal verification techniques, simulation remains the workhorse for system analysis. A plethora of simulation tools are available today, from academia as well as from industry. These tools support a large variety of modeling languages, targeted at different types of systems from various disciplines (e.g., mechanical, electrical, digital, continuous or discrete, or mixes thereof). Unfortunately, these tools can rarely interoperate. This is a problem because modern cyber-physical systems are highly complex and multidisciplinary, requiring specialized modeling languages and tools from several domains.

The *Functional Mock-up Interface*¹ (FMI) is a standard developed to address this problem. FMI defines an XML schema for describing simulation components and a C API that these components must implement. The components are called *functional mock-up units*,

or FMUs. An FMU is typically generated automatically (*exported*) from some simulation tool, and corresponds to a (sub-)model designed in that tool. The submodels/FMUs are then *imported* into a *host* simulator. The host commands the simulation by calling the API methods of the FMUs, thus effectively achieving integration of the original simulation environments. FMI supports two integration modes: (a) *model exchange*, where the host simulator handles the numerical integration; and (b) *co-simulation*, where each FMU implements its own numerical integration mechanism (or any other internal mechanism to advance its state in time). Because each mode imposes its own requirements on FMUs (for instance, in model exchange, the FMUs must provide the host with information such as state derivatives, which are not necessary for co-simulation) the FMI APIs for the two modes are different.

In this paper, we use FMI in order to connect two state-of-the-art modeling and verification tools for cyber-physical systems: SPACEEX (Frehse et al., 2011) and UPPAAL (Larsen et al., 1997). SPACEEX is a tool for modeling and verifying *hybrid systems* (Alur et al., 1995). UPPAAL is primarily a model-checker for *timed automata* (Alur and Dill, 1994), however, it also supports statistical model-checking of hybrid systems (David et al., 2011).

Our goal is to integrate these two tools for co-simulation. That is, we want to be able to: (a) build a sub-model of the system (e.g., the model of the *plant* under control) in SPACEEX; (b) build another sub-model (e.g., the *controller*) in UPPAAL; (c) automatically generate an FMU for each sub-model; (d) import the FMUs, connect and co-simulate them in a host environment.

The motivations for connecting SPACEEX and UPPAAL in this manner are numerous. First, although both SPACEEX and UPPAAL support simulation of hybrid systems, each tool offers its own modeling language, which is not compatible with that of the other tool. Translating from one language to the other is limited to common features supported by the tools. For example, even though the frameworks CIF (Agut et al.,

¹See <https://www.fmi-standard.org/> for more details.

2013; Beohar et al., 2010) and HSIF (Pinto et al., 2006) solve the complexity problem of one format translation to another by performing at most two translations, the approach still suffers from the fact that UPPAAL features like committed locations and C-like function code are not supported in SPACEEX and UPPAAL has limited support for ODEs. Moreover, by using co-simulation, we are able to take advantage not just of the specific strengths of the language of each tool, but also of their native simulation engines, since each FMU is internally running essentially a “copy” of the simulation algorithm of the original tool.

As host environment we use the tool Ptolemy². Ptolemy is a modeling and simulation environment for heterogenous systems (Eker et al., 2003). Recently, support has been implemented in Ptolemy for using it as a host environment for co-simulation based on FMI. FMUs (developed by other tools) can be imported into Ptolemy, connected using Ptolemy’s graphical user interface, and co-simulated using an implementation of the co-simulation algorithm described by Broman et al. (2013). This algorithm has desirable properties, such as *determinacy*, namely, the fact that the results of the simulation are independent of arbitrary factors such as names of the FMUs, order of creation, or order of evaluation in the diagram.

The contributions of this paper are the following:

1. We show how FMUs can be generated automatically from models of hybrid and timed automata built in SPACEEX and UPPAAL. There are several subtleties involved in this, as hybrid and timed automata are models designed primarily with verification in mind, whereas FMI is designed for simulation and therefore imposes certain properties on FMUs, such as determinism.
2. We report on the implementation and case studies. In particular, we apply our co-simulation framework to a room heating benchmark (Fehnker and Ivancic, 2004).
3. We validate the co-simulation algorithm proposed by Broman et al. (2013) by comparing the results of the case study in two settings: (a) when the case study is modeled and simulated in a single tool, and (b) when the various components of the case study are modeled in two tools and co-simulated using our framework. We show that our co-simulation framework computes the same simulation trajectories as the setting (b) provided that the maximum simulation step size of co-simulation is sufficiently small.
4. We demonstrate how stochastic simulations can be included into the composite model with hybrid systems and applied a simple statistical measurement

to show the potential for statistical model checking using FMI co-simulations.

The rest of the paper is organized as follows. In Sec. 2, we introduce the necessary background on FMI for this work. Afterwards, we present our translation of SPACEEX and UPPAAL models into FMUs in Sec. 3. This is followed by the case study in Sec. 4. We discuss related work in Sec. 5. Finally, we conclude the paper in Sec. 6.

2 Background on FMI

Conceptually an FMU can be seen as a (timed) state machine. This machine has a set of input variables (or *ports*), a set of output variables, and a set of internal states. The machine interacts with its environment only by means of a clearly defined set of *interface methods*. These methods are specified in the FMI standard. For the purposes of this paper, and following the formalization presented by Broman et al. (2013), the key interface methods of FMI (for co-simulation) are:

- A method to initialize the state of the FMU. If S is the set of states of the FMU, then `init` $\in S$.
- A method `set` to set a given input variable to a certain value. The signature of `set` is `set` : $S \times U \times \mathbb{V} \rightarrow S$, where U is the set of input variables of the FMU, and \mathbb{V} is the set of all possible values (for simplicity we ignore typing and use a single universe \mathbb{V} of values for all variables). Given state s , input variable $u \in U$, and value $v \in \mathbb{V}$, `set`(s, u, v) returns the new state obtained after setting u to v .
- A method `get` which returns the value of a given output variable. Its signature is `get` : $S \times Y \rightarrow \mathbb{V}$, where Y is the set of output variables of the FMU. Given state s and output variable $y \in Y$, `get`(s, y) returns the value of y in s .
- A method `doStep` which advances the state of the machine in time. Its signature is `doStep` : $S \times \mathbb{R}_{\geq 0} \rightarrow S \times \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers. The behavior of `doStep` is explained below.

As said above, an FMU is essentially a state machine: the `get` method corresponds to the output function of the machine, while the `doStep` method corresponds to the transition function. The difference is that `doStep` takes as input a *time step* $h \in \mathbb{R}_{\geq 0}$: in that sense, an FMU is a timed state machine.

The behavior of `doStep` is as follows. Given state $s \in S$, and time step $h \in \mathbb{R}_{\geq 0}$, a call to `doStep`(s, h) is interpreted as the co-simulation algorithm “asking” the FMU to perform a simulation step of length h . For a

²See <http://ptolemy.eecs.berkeley.edu/>.

number of reasons, including numerical integration issues, the FMU may “accept” or “reject” this request. If it rejects, it means that it was not able to advance time by h (but may have been able to advance time by a smaller delay $h' < h$). Formally, $\text{doStep}(s, h)$ returns a pair (s', h') where $s' \in S$ is a state and $h' \in \mathbb{R}_{\geq 0}$ is a time step, such that:

- either $h' = h$, which is interpreted as F having *accepted* h , and having moved to a new state s' ;
- or $0 \leq h' < h$, which is interpreted as F having *rejected* h , but having made partial progress up to h' , and having reached a new state s' .

It is worth noting that FMUs are *deterministic* machines, in the sense that for a given sequence of inputs (i.e., a sequence of input values and time steps), the sequence of states and outputs that the machine produces is unique. This is because there is a unique initial state $\text{init} \in S$, and set , get , doStep are all *total functions*. Moreover, the fact that these functions are total implies that the machine is able to accept any input at any time, therefore, it is implicitly *input-enabled*.

We also rely on zero-time steps in a sense of allowing $\text{doStep}(s, h)$ calls with $h = 0$ (despite that version 2.0 of the FMI standard forbids this), because they are essential for modeling discrete transitions like instantaneous mode switches in hybrid automata models.

In addition to the above, each FMU comes with a set of *input-output dependencies*, $D \subseteq U \times Y$. D specifies for each output variable which input variables it depends upon (if any): $(u, y) \in D$ means that output variable y depends on input variable u . This information is used to ensure that a network of FMUs has no cyclic dependencies, and also to determine the order in which all network values are computed during a simulation step (Broman et al., 2013).

FMI specifies the methods that every FMU must implement, but it does *not* specify the co-simulation algorithm (also called a *master algorithm*). In fact, devising such an algorithm with good properties is not a trivial problem, and has been the topic of previous work (Broman et al., 2013). In that work, two co-simulation algorithms were proposed and proved to have desirable properties, such as termination of a simulation step, and *determinacy*. The determinacy property says that the results of a simulation do not depend on the order in which the algorithm chooses to call doStep over a set of FMUs. This ensures that the simulation results are well-defined and are not influenced by arbitrary factors such as FMU names, order of creation, geometrical position in the diagram of a graphical model, etc., as is often the case with simulation tools.

In a nutshell, the co-simulation method proposed by Broman et al. (2013) relies on the following principle. First, the co-simulation algorithm chooses a default time step, h_{\max} , called the *maximum step size*. Second,

the algorithm saves the state of each FMU in the model (FMI specifies methods for an FMU to export and import its state, although these are optional). Assuming there are n FMUs, F_1, \dots, F_n , the algorithm maintains n states, s_1, \dots, s_n . Third, the algorithm calls $F_i.\text{doStep}(s_i, h_{\max})$ on each FMU F_i , and collects the returned time steps h'_1, \dots, h'_n . There are two cases: either all FMUs accepted the proposed time step, i.e., $h'_1 = h'_2 = \dots = h'_n = h_{\max}$, in which case this simulation step is over, and the algorithm proceeds to the next one; or at least one FMU F_i rejected h_{\max} , i.e., $h'_i < h_{\max}$ for some i . In the latter case, the algorithm computes the minimum of h'_1, \dots, h'_n , $h_{\min} = \min\{h'_1, \dots, h'_n\}$, restores the saved state of each FMU, and tries again with new step size h_{\min} .

Assuming that the FMUs satisfy the reasonable “monotonicity” property that if they were able to advance time by h'_i then they are also able to advance time by any smaller step, and by the fact that h_{\min} is smaller than all h'_i , the second attempt is guaranteed to succeed. That is, h_{\min} will be accepted by all FMUs. As a result, at most after two attempts, a co-simulation step is successful, and the algorithm proceeds with the next step, repeating the same procedure as above.

The FMI standard sets out a framework where FMUs share the notion of time and exchange variable values via input-output ports: outputs from one FMU are mapped as inputs to other FMU(s) and so on. The output port values are said to be owned and controlled by the emitting FMU, whereas the inputs are computed and provided by another (outputting) FMU. The framework foresees that before producing an output an FMU may first need some input values and thus input-output dependency information is introduced. Overall the I/O port connectivity graph derived from the model of interconnected FMUs, together with the local I/O dependencies of each individual FMU, result in a global I/O dependency graph for the entire model (Broman et al., 2013).

Time and I/O values are synchronized by the co-simulation algorithm: the time is agreed by repeatedly consulting each FMU and the I/O values are propagated according to dependencies. The co-simulation algorithm assumes that each FMU provides a static dependency list of its ports before simulation starts, and that the resulting global I/O dependency graph is acyclic, and therefore there exists a schedule for computing the value of every input port before the value of a dependent output port is requested (Broman et al., 2013).

3 Translating Models into FMUs

The behavior of individual FMUs is provided by the model-checker’s simulation engines based on the guidelines described by Tripakis (2015). In particular, the report distinguishes continuous and discrete dynamics. The continuous behavior is modeled by differential equations over continuous variables whose values can be

shared among FMUs by the means of port connections. The output ports of an FMU are mapped to the owned/controlled variables which are read and written to, whereas input ports map to read-only variables within the FMU.

The discrete behavior is modeled by discrete transitions in the timed/hybrid automata control flow structure. The discrete transitions are designed to be executed with micro-steps of zero delay. Transitions can also be decorated with event labels and each tool supports its own kind(s) of synchronizing compositions internally and therefore the discrete transition synchronization is also handled individually within the tools. Tripakis (2015) provides the means of discrete transition synchronization by allocating two special port variables: one for incoming (input) synchronization and one for outgoing (output) synchronization. The domain of discrete input (output) ports coincides with the set of input (output) labels plus a special value *absent* which denotes no synchronization or an internal discrete transition.

3.1 Uppaal

UPPAAL uses timed automata models (Alur and Dill, 1994), extended with discrete variables over structured types to describe behaviors of a timed system. In timed automata, the continuous dynamics is controlled by real-valued clock variables (with derivatives set to one) and discrete states complemented with integer variables – both of which are candidates for exchange via FMU input-output ports. Statistical model checking (SMC) extensions (David et al., 2011, 2015) allow a finer control of the clock derivatives by means of ordinary differential equations, moreover the discrete transitions are stochastic where the execution is determined by probability distributions over time and over branching edges. The stochastic semantics of a parallel composition is similar to the FMI co-simulation algorithm (Broman et al., 2013): the way the minimum delay is negotiated and thus the timed composition within the FMI framework is straightforward, and task is to find a systematic way of handling discrete synchronizations. UPPAAL also supports the maximal progress or ASAP semantics on edges labeled with urgent channels.

UPPAAL supports the notion of discrete I/O synchronization natively by means of input and output channel labels. Thus, its discrete input and output transitions can be mapped directly to the input/output port variables of an FMU that is dedicated to transfer the synchronization label name. Nonetheless, we distinguish the following kinds of transitions: internal (transitions without I/O channel synchronization or internally synchronized transitions for which channels are not marked as input or output), input transitions (labeled by an input synchronization where the channel name is marked as an FMU input), and output transitions (labeled by an output synchronization where channel is marked as an FMU out-

put). The marked outputs are controlled by the UPPAAL simulation and are executed asynchronously irrespective of whether the receiving FMU is ready to synchronize. Meanwhile, the input transitions are executed only when there is a corresponding input label set on a discrete input port. At most one (internal, input or output) transition is allowed at a time, hence fine-grained simulation control can be achieved by the co-simulation algorithm.

UPPAAL FMUs do not introduce I/O dependencies between continuous variables because the models do not use algebraic expressions to compute variable values. Instead of algebraic expressions the automata use discrete transitions to update the variable values. However, only one discrete transition is allowed at a time, therefore all discrete outputs have dependencies on the inputs dedicated to synchronization labels which restrict the selection of a particular discrete transition and hence specific variable update.

3.2 SpaceEx

SPACEEX (Frehse et al., 2011) uses hybrid automata to describe system behavior where the continuous variable derivatives are constrained by differential equations. The continuous variables are candidates for input and output exchange via FMU ports. The discrete transitions of hybrid automata can be decorated with labels. Synchronization may involve multiple participating processes, but there is no notion of input and output – all processes are equal contributors, therefore the simulator needs to implement the input/output semantics required by FMI. We use a special label naming notation to mark input and output labels (see Fig. 6). The transitions with input labels are only executed when the discrete input variable of FMU is set to the corresponding label name. Meanwhile, the transitions with an output label are controlled by SPACEEX' simulation, and are executed asynchronously by setting the discrete output variable with the label name irrespective of whether the receiving FMU can synchronize with it. We ensure the SPACEEX FMU determinism by enforcing the must-semantics of discrete transitions in a hybrid automaton. In other words, a discrete transition is taken as soon as its guard is enabled. Finally, we resolve the non-determinism between input, output, and internal transitions in the following way: input transitions have priority over output transitions and output transitions are preferred over the internal ones.

Both UPPAAL and SPACEEX translations simulate the source models as they are without intermediate transformations, except of the following additions: 1) input enabledness is ensured by broadcast channels in UPPAAL modeling and asynchronous I/O is implemented for SPACEEX synchronization labels, 2) for determinization SPACEEX uses maximal progress whereas UPPAAL uses stochastic semantics with a possibility of urgent channels for maximal progress.

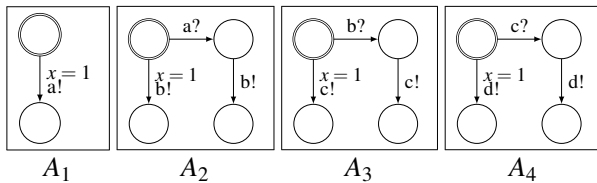


Figure 1. An example of four timed automata chain.

3.3 Discussion on Co-Simulation Semantics

In this section, we discuss the co-simulation semantics and contrast it to those typically used by a model-checking tool. In particular, we demonstrate by example how the FMI co-simulation algorithm resolves input/output dependencies and contrast it with execution analysed in a model checker. Our goal is to offer insights in the differences of the two semantics.

Consider a system model shown in Fig. 1 which consists of four timed automata composed in parallel. Labels of the form $a!$ denote sending output a , whereas $a?$ denotes receiving an input a . The variable x is a clock measuring time starting from zero. The constraint $x = 1$ is a guard which allows the corresponding transition of the automaton to occur only if the guard is satisfied, i.e., in this case only when x equals 1. The automata synchronize in a chain: the first can output a to the second one, the second one can output b to the third one and so on.

In principle, the system can be loaded into an FMI model in any combination: individually (one automaton per FMU) or collectively (multiple automata per FMU), but before an FMU can be loaded into an FMI model, it must declare its input/output dependencies. According to Broman et al. (2013) each automaton should expose an input/output variable which will contain the synchronization label value. Automaton A_1 in the example above will have only an output variable, which may have values $\{a, absent\}$. Automaton A_2 will have an input variable ranging over $\{a, absent\}$ and an output variable ranging over $\{b, absent\}$, and so on. The special value *absent* denotes that currently there is no synchronization. Timed automata must declare a dependency between its input and output label variable in order to avoid simultaneous input and output synchronizations.

In addition, it is assumed that each FMU is *input-enabled*, meaning that it can handle (i.e., it is able to receive) any declared input at any time. If a component is not input-enabled and an input synchronization is triggered then simulation is aborted, to avoid such situation we allow only broadcast channels, which do not block the sender process and receiver may simply ignore the synchronization if has no receiving edge.

Suppose the automata from Fig. 1 are loaded within separate FMUs and connected according to synchronization labels. That is, the output of $FMU(A_1)$ is connected to the input of $FMU(A_2)$, the output of $FMU(A_2)$ is connected to the input of $FMU(A_3)$, and so on. The co-simulation algorithm would detect that it has to fulfill inputs values for the $FMU(A_4)$, $FMU(A_3)$, and $FMU(A_2)$

in order to proceed, therefore the input/output value propagation will have to start with $FMU(A_1)$ and then proceed to the $FMU(A_2)$ etc.. Once the values of all input and output variables are propagated, the algorithm proceeds with advancing each FMU in time by calling $doStep()$. It is this dynamic behavior in time which interests us in this example.

In particular, observe that $A_{2,3,4}$ automata are non-deterministic in the sense that, according to UPPAAL semantics, at time $x = 1$ an automaton can either delay, or take an outputting transition, or synchronize on inputs. For instance, at time $x = 1$, A_2 can either emit b , or receive a (which will be available in this case, because it is sent by A_1 at exactly that time), or let the time pass. In timed automata semantics, all these options are possible at the individual component level. Moreover, not only individual components can be non-deterministic, but their composition is non-deterministic as well, based on so-called *interleaving semantics*. This means that when multiple automata are enabled at a given time, the choice of which one to execute is arbitrary. Non-determinism is a useful abstraction and thus model reduction technique in verification and model-checking. The same is true when these tools are used for simulation, i.e. different simulations in UPPAAL may yield different results.

In FMI, the situation is very different, as all FMUs are treated as deterministic components, and their composition, ensured by the co-simulation algorithm, is guaranteed to yield deterministic results as well. Interestingly, in this example, if all automata decide to output at time $x = 1$, some of them will succeed outputting in parallel, while others will be preempted by incoming inputs. In particular, the master algorithm will request $FMU(A_1)$ to produce its output, and thus $FMU(A_2)$ will be busy handling an input and will not be producing output at that time. Since $FMU(A_2)$ is not sending anything, then $FMU(A_3)$ will be free to produce an output and hence preempt $FMU(A_4)$.

As witnessed from above, such FMI system selects a particular sequence of steps (which is expected) but is not able to simulate all possible execution orders as in original semantics even if we allow FMUs to determinize their actions by themselves, which means that FMI simulations are selecting a particular subset of all possible behaviors and some behaviors may not be reproducible in FMI. Also FMI simulations may contain parallel synchronizations (e.g. actions $A_1 \xrightarrow{a} A_2$ and $A_3 \xrightarrow{c} A_4$ at the same computation step) which are possible only in several steps in timed automata semantics (action a and only then action c within zero-time), hence the intermediate state between a and c actions might not be accessible in FMI without very fine grained control over individual $doStep()$ calls in one zero-time computation step. However, the successor state of such parallel executions can be matched with a state after multiple transitions in the given automata semantics, hence the FMI simulation

states in between system computation steps are included in the original semantics, albeit definite proof requires more formal insight to examine all scenarios.

4 Case Study

We have implemented the FMI standard in the UPPAAL (Larsen et al., 1997) and SPACEEX (Frehse et al., 2011) model checkers by providing model export to FMU³. In this section, we present and evaluate the performance of the resulting FMI framework on a case study inspired by the well-known room heating benchmark originally proposed by Fehnker and Ivancic (2004). Our model consists of a room with a heater (Fig. 2) and a controller (Fig. 3) which regulates the heater behavior. We model the room and the controller as a SPACEEX and UPPAAL FMU, respectively (see Fig. 4). Our bang-bang controller turns the heater on and off as soon as some temperature thresholds T_{low} and T_{high} have been reached. The as-soon-as-possible behavior is enforced by using urgent channels which effectively make the controller deterministic. The room temperature T evolves according to the following differential equation:

$$\begin{aligned}\dot{T} &= k \cdot (T_{env} - T) + h_{power} \\ \dot{T}_{env} &= 0 \\ \dot{h}_{power} &= 0\end{aligned}$$

In other words, the room temperature depends linearly on the difference between the current room temperature T and outside temperature T_{env} . We assume the outside temperature T_{env} and heater power h_{power} to be constant. The constant k defines the heat exchange rate between the room and outside environment. If the heater is off, the heater power is set to zero.

4.1 Evaluation

We evaluate our FMU framework by comparing simulation trajectories of the FMUs with the ones produced by a SPACEEX model consisting of both the controller and room components. We consider three different simulation step values: 1 (see Fig. 5a), 0.1 (see Fig. 5b) and 0.01 (see Fig. 5c). Considering the simulations, we observe that the FMU trajectories *overshoot* the controller constraints in the sense that the controller exhibits a delayed reaction when the room temperature crosses the temperature thresholds. The behavior is justified by the fact that the method call `doStep` for every FMU relies only on the *local* information about the state evolution when making decisions, e.g., the controller FMU does not have any information about the room temperature evolution beyond the value which can be provided when

³A package containing the benchmarks is available for download at <http://swt.informatik.uni-freiburg.de/tool/spaceex/co-simulation>.

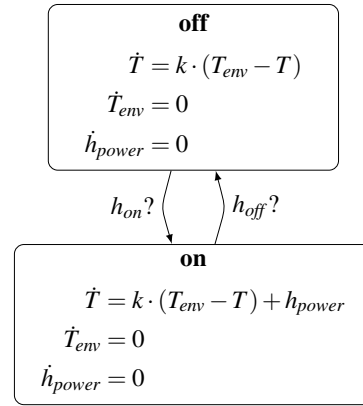


Figure 2. Room component modelled in SPACEEX. The component switches between “on” and “off” modes. The temperature variable T is exported as output and synchronizations labels h_{on} and h_{off} as inputs.

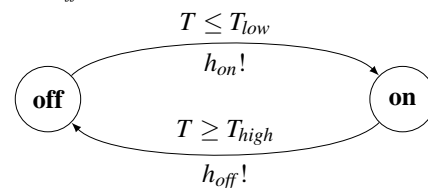


Figure 3. Controller in UPPAAL uses urgent channels to ensure as-soon-as-possible transition trigger. Temperature T is an input and labels h_{on} and h_{off} are outputs.

the method `doStep` is called. Therefore, the controller FMU detects that the guard is enabled only *simulation iteration later* after this event has already happened. We observe that the impact of the overshooting can be made arbitrary small by choosing a small enough simulation step (see Fig. 5c vs. Fig. 5a and Fig. 5b).

We note that the overshooting problem is *inherent* to the considered master algorithm and can be circumvented by incorporating additional cross-component knowledge into the master algorithm. Overall, our experiments validate that on this case study our co-simulation framework based on SPACEEX and UPPAAL provides equivalent simulation results compared to the setting where all components are modelled in one tool.

4.2 Supervisory Control Example

In this section, we show how supervisory control systems similar to the benchmarks presented by Fehnker and Ivancic (2004) can be modeled using the FMI paradigm. Compared to Section 4.1, we consider a model of the building with two rooms sharing a common wall and a heater. In this setting, the room temperature is influ-

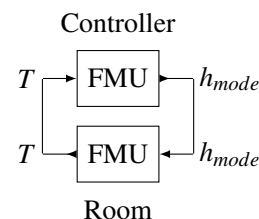


Figure 4. SPACEEX and UPPAAL FMUs connected using the room temperature T and heater mode h_{mode} .

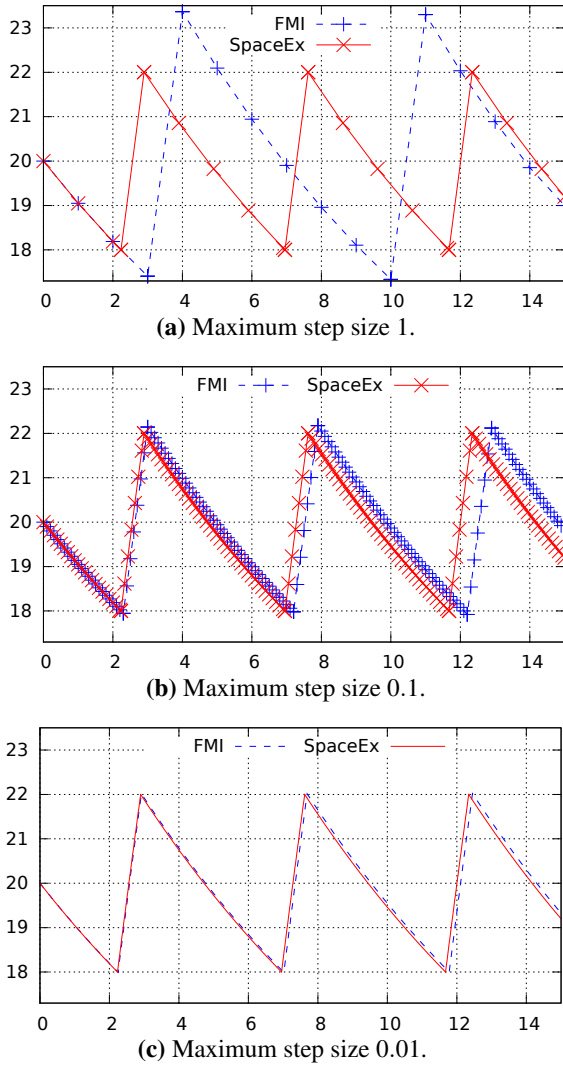


Figure 5. Simulation trajectories: each red \times is a data point reported by SPACEEX, and blue $+$ reported by the co-simulation.

enced by both the outside temperature and heat transfer between the rooms. Figure 6 shows a hybrid automaton from SPACEEX modeling the room temperature dynamics. The difference from the previous example here is an extra term $(T_{other} - t) * 0.2$ denoting a contribution from another room. Another room is modeled analogously except that it responds to *heater2_on* and *heater2_off* signals instead of *heater1_on* and *heater1_off*.

Our controller consists of two parts: local bang-bang controller and a supervisor shown in Fig. 7. In order to model the transitions of the heaters between the rooms, we assume that the controllers can be turned on/off by the supervising controller. Therefore, the local controller has an extra mode besides *On* and *Off* which stands for the controller being currently deactivated. The supervising controller has two kinds of stochastic behavior: it can pick any pair of rooms (one recipient and another donor) to transfer the heater, and it can choose the timing of transfer. When a pair of rooms is selected (by choosing concrete room identifiers for *rec* and *donor* variables) the donor is disabled by moving from location *decide* to lo-

cation *move* and the recipient is enabled by going from *move* to *idle*. The supervisor may stay in location *idle* arbitrary long, but the exact duration is decided by an exponential probability distribution of rate 1 which means the duration of $1/1$ time units on average. Similarly the supervisor may stay in *decide* and *move* but the duration will be $1/10000$ on average, i.e. denoting that the heater is moved rather quickly.

Figure 8 shows the overall component connectivity diagram where the supervisor is reading temperatures from each room and controls the local movable heater controllers. The movable heaters then may either turn on the heat in their room or let them cool off giving the heat to

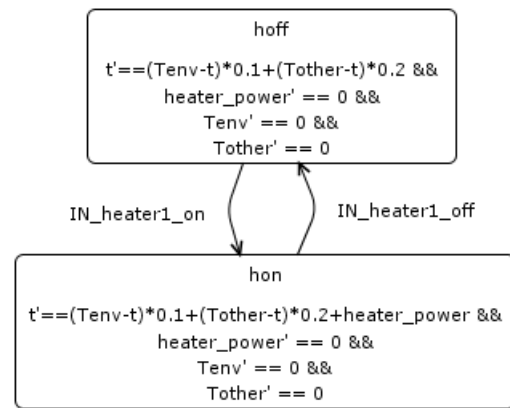
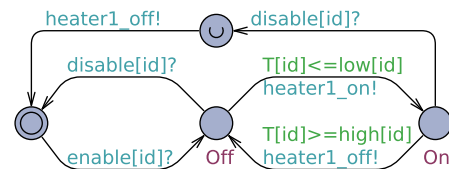
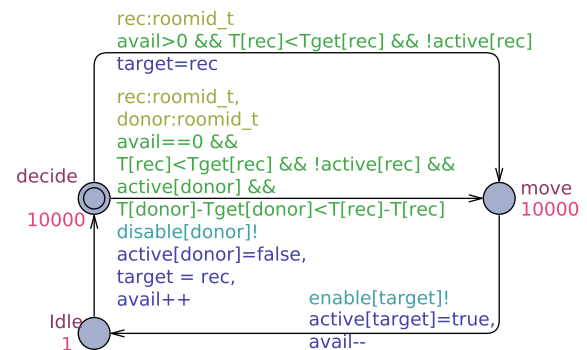


Figure 6. Hybrid automaton for a heated room connected to another room. Inputs are temperatures T_{env} , T_{other} and labels *IN_heater1_on* and *IN_heater1_off*, while output is temperature t . We use the prefix *IN* to mark input labels.



(a) Local bang-bang controller which can be moved (disabled). The inscribed U means urgent location where time delay is not allowed. The inputs are temperature variable $T[id]$ and labels *enable[id]* and *disable[id]*, while outputs are labels *heater1_on* and *heater1_off*.



(b) Supervising controller moves the heaters between rooms by reading inputs on $T[i]$ and sending outputs on labels *enable[i]* and *disable[i]* where i is the room index.

Figure 7. Two layers of UPPAAL controllers.

outside. The individual heated rooms are then connected to the outside temperature and to each other denoting the heat exchange. The splitter FMUs are repeaters needed to connect multiple components to the same signal.

In the following, we discuss the behavior of the resulting composed model. Figure 9 shows the temperature dynamics in each room. In particular, the plot shows that in the beginning the temperature drops until the supervisor detects a room temperature below $T_{get} = 17^\circ$, then around 6 time units a heater raises the temperature in room 1. The local controller keeps rising the temperature until it goes over 22° bound at around 7.5 time units. Notice that the temperature in room 2 also rises due to heat exchange between the rooms. Around 10 time units the supervisor decides to hand over the heater to room 2. At 14 time units the heater is switched back to room 1 and so on. We can conclude that even though the temperature drops well below 18° overall it seems that the controllers manage to sustain the temperature at the similar level without losing control (without dropping to outside temperature level).

4.3 Stochastic Simulations and SMC

The following is a demonstration of statistical model checking (SMC) using the FMI framework. We show how the performance of two stochastic controllers simulated by UPPAAL can be compared using SMC approach together with the heated room simulation provided by SPACEEX. Figure 10 shows two controllers: (a) reacting within 1 time unit to 18.0° and 22.0° temperature bounds and (b) reacting within 2 time units to 19.0° and 21.0° temperature bounds. The channels used in these controllers are not urgent and therefore the delay between temperature detection and heater activation is decided stochastically based on uniform distribution over the allowed delay by invariants, i.e. the concrete delay will be chosen from $[0, 1]$ for the first controller and from $[0, 2]$ for the second one. The *On* and *Off* locations do not have any invariant and therefore in principle the process may stay there forever. In such cases UPPAAL uses an exponential (Poisson) probability distribution to decide a particular time delay and hence asks to provide a rate of the exponential. The higher the exponential rate, the shorter the delays, hence we can provide a high rate to ensure that the detecting transition is fired arbitrary quickly.

In our setup, we would like to know which controller is better at keeping the room temperature within 18.0° and 22.0° bounds. In order to answer this question we setup two FMI models for each controller with an equal room, run 100 simulations with 100 time units in length and 0.05 granularity, compute the amount of time spent outside the temperature range for each simulation and then compute the confidence intervals for both models. Table 1 shows a summary of amounts of time during which the temperature was either below or above the range. The estimated time duration use confidence in-

terval (CI) notation which means that if we repeat the measurement experiment then the real mean (which is unknown) will fall into the interval with a probability of 95%. The results show that the second controller was more successful at maintaining the lower bound of the temperature, but was more overshooting beyond the upper bound. In total, the first controller kept the temperature in good range longer by 8.57 time units on average, which is much larger than confidence interval, hence the first controller is better.

Table 1. Time with temperature outside the range (95% CI).

Controller	Time below	Time above	Total
Wide and fast	7.56 ± 0.20	32.69 ± 3.36	40.26 ± 0.59
Narrow and slow	2.40 ± 0.19	46.43 ± 0.82	48.83 ± 0.79

5 Related Work

The FMI standard and corresponding documentation are constantly evolving, as new versions of the standard are developed. The web site⁴ also contains a list of tools supporting FMI. Descriptions of FMI can also be found in the academic literature (Blochwitz et al., 2011).

Discussions about the limitations of FMI can be found in the works by Broman et al. (2013, 2015). Broman et al. (2013) also formalize the main methods of FMI (*get*, *set*, *doStep*) by establishing a *contract* (pre/post-conditions) for each method and propose a *master algorithm* (i.e., a co-simulation algorithm). Furthermore, the authors proves its termination, determinacy, and other properties. However, the paper does not discuss how FMUs can be created. A different, master-slave based, co-simulation approach is proposed by Bastian et al. (2011), but formal properties such as determinacy are not discussed in this work.

Broman et al. (2015) defines a suite of test models that should be supported by a hybrid co-simulation environment, giving a mathematical model of an ideal behavior, plus a discussion of practical implementation considerations. Furthermore, the paper describes a set of basic modeling components in the spirit of Ptolemy actors (constant, gain, adder, integrator, etc.). Finally, the authors provide a kind of denotational description for each component (input and output signals), but no encoding into FMUs is discussed.

The FMU generation problem for various formalisms is discussed by Tripakis (2015). This work only refers to a generic model of timed machines which does not include the particularities of UPPAAL's timed automata. In addition, hybrid automata are not considered in this work.

Recently, the co-simulation algorithm presented by Broman et al. (2013) has been implemented in the

⁴<https://www.fmi-standard.org/>

how the non-deterministic models can be determinized using stochastic semantics and included into FMI co-simulation. We also provided an example how statistical model checking can be performed using numerous FMI simulations which is an essential feature evaluating stochastic behavior. The integration of model-checkers into co-simulation frameworks provides further possibilities of analyzing early design models like conformance monitoring by checking that a simulation trace of a refined (e.g. hybrid) model is included in a more abstract (e.g. timed automata) specification. We envision our work being a further step towards integrating tools developed in the formal methods community into the industrial system design and modeling workflow of cyber-physical systems.

7 Acknowledgments

We are grateful to Christopher Brooks, Fabio Cremona, and Edward Lee from UC Berkeley, for their work on the Ptolemy framework. This work was partly supported by the European Research Council (ERC) under grant 267989 (QUAREM), by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS), by EU FET project SENSATION, the Sino-Danish Center IDEA4CPS and the Center DiCyPS of the Danish Innovation Foundation, by Academy of Finland, by the National Science Foundation (awards #1329759 and #1139138), and by the Industrial Cyber-Physical Systems Research Center (iCyPhy) supported by IBM and United Technologies Corporations.

References

- D.E. Nadales Agut, Dirk A. van Beek, and J.E. Rooda. Syntax and semantics of the compositional interchange format for hybrid systems. *The Journal of Logic and Algebraic Programming*, 82(1):1 – 52, 2013. ISSN 1567-8326. doi:10.1016/j.jlap.2012.07.001.
- R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
- Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. HYST: a source transformation and translation tool for hybrid automaton models. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC, Seattle, WA, USA, April 14-16, 2015*, pages 128–133. ACM, 2015.
- Jens Bastian, Christoph Clauß, Susann Wolf, and Peter Schneider. Master for Co-Simulation Using FMI. In *8th International Modelica Conference*, 2011.
- Harsh Beohar, D. E. Nadales Agut, Dirk A. van Beek, and Pieter J. L. Cuijpers. Hierarchical states in the compositional interchange format. In Luca Aceto and Pawel Sobocinski, editors, *Proceedings Seventh Workshop on Structural Operational Semantics, SOS 2010, Paris, France, 30 August 2010.*, volume 32 of *EPTCS*, pages 42–56, 2010. doi:10.4204/EPTCS.32.4.
- 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 International Modelica Conference*, Dresden, Germany, March 2011. Modelica Association.
- D. Broman, C. Brooks, L. Greenberg, E. A. Lee, S. Tripakis, M. Wetter, and M. Masin. Determinate Composition of FMUs for Co-Simulation. In *13th ACM & IEEE International Conference on Embedded Software (EMSOFT’13)*, 2013.
- D. Broman, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. Requirements for Hybrid Cosimulation Standards. In *Hybrid Systems: Computation and Control (HSCC)*, 2015.
- Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical model checking for networks of priced timed automata. In Uli Fahrenberg and Stavros Tripakis, editors, *Formal Modeling and Analysis of Timed Systems*, volume 6919 of *Lecture Notes in Computer Science*, pages 80–96. Springer Berlin Heidelberg, 2011.
- Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *International Journal on Software Tools for Technology Transfer*, 2015.
- J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neundorffer, S. Sachs, and Y. Xiong. Taming heterogeneity – the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.
- Ansgar Fehnker and Franjo Ivancic. Benchmarks for hybrid systems verification. In *In Hybrid Systems: Computation and Control (HSCC)*, pages 326–341. Springer, 2004.
- Y. A. Feldman, L. Greenberg, and E. Palachi. Simulating Rhapsody SysML Blocks in Hybrid Models with FMI. In *10th Modelica Conference*, pages 43–52, 2014.
- Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceX: Scalable Verification of Hybrid Systems. In Shaz Qadeer Ganesh Gopalakrishnan, editor, *23rd International Conference on Computer Aided Verification (CAV)*, LNCS. Springer, 2011.
- Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

Alessandro Pinto, Alberto L. Sangiovanni-Vincentelli, Luca P. Carloni, and Roberto Passerone. Interchange formats for hybrid systems: review and proposal. In *Hybrid Systems: Computation and Control*, HSCC. Springer, 2005.

Alessandro Pinto, Luca P. Carloni, Roberto Passerone, and Alberto Sangiovanni-Vincentelli. Interchange format for hybrid systems: Abstract semantics. In Joao P. Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control*, volume 3927 of *LNCS*, pages 491–506. Springer Berlin Heidelberg, 2006.

Uwe Pohlmann, Wilhelm Schäfer, Hendrik Reddehase, Jens Röckemann, and Robert Wagner. Generating Functional Mockup Units from Software Specifications. In *9th Modelica Conference*, pages 765–774, 2012.

Stavros Tripakis. Bridging the Semantic Gap Between Heterogeneous Modeling Formalisms and FMI. In *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation – SAMOS XV*, 2015.

Automated Deployment of Modelica Models in Excel via Functional Mockup Interface and Integration with modeFRONTIER

John Batteh¹ Jesse Gohl¹ Anand Pitchaikani¹
Alexander Duggan² Nader Fateh²

¹Modelon Inc., USA, {john.batteh, jesse.gohl, anand.pitchaikani}@modelon.com

²ESTECO North America Inc., USA, {duggan, fateh}@esteco.com

Abstract

This paper describes a method for automated deployment of Modelica models as simulators in Microsoft Excel using Functional Mockup Interface (FMI) and FMI Add-in for Excel. Using existing interfaces, integration with modeFRONTIER is demonstrated and illustrated with several different example models in different physical domains to highlight the range of applications and types of analyses that can be covered with the automated toolchain. This toolchain can be applied to any FMU and streamlined with automation enabled by the supporting annotations.

Keywords: Modelica, Functional Mockup Interface, automation, simulator, optimization, robust design, Microsoft Excel

1 Introduction

Model-based methods for development of physical and control systems have been applied across engineering domains to streamline development, reduce time to market, and manage cost and innovation. As integrated systems become increasingly complex with multi-domain interactions spanning a range of disciplines, the role of virtual models and analysis techniques in the product development process continues to grow in importance.

To meet the demand for increased model-based engineering, the ability to efficiently develop and deploy models across an enterprise is a key enabler. Models are no longer handled only by domain experts in CAE departments but are being deployed to engineers who may not have intimate knowledge of the underlying models but still are required to use models effectively to support engineering processes. With the proliferation of models throughout the enterprise, the desire for simulators outside of the original model development environment is natural and a key enabler for increased acceptance and usage of models. While it is clearly in the best interest of model users to receive models in a format of their choosing, this desire requires careful balancing of the time and effort

required to deploy the simulators, typically time spent by highly-skilled and resource-constrained model developers. Automated simulator deployment can certainly help bridge the gap between the model development and deployed simulator environments.

Open standards such as the Modelica modeling language and Functional Mockup Interface (FMI) for model exchange and co-simulation can streamline the modeling and deployment process by providing standard, non-proprietary interfaces between tools. In addition to the ability to share and integrate models from a variety of tools as FMUs, the FMI co-simulation standard provides a convenient way to deploy models outside of the original development environment as simulators. FMI-based simulators are increasingly common and rapidly gaining acceptance across industries due to the flexibility offered in simulation platforms, IP protection, and also due to the potential for flexible licensing of the deployed simulators. While FMI capability exists in nearly every Modelica-based modeling platform, the rapid adoption of FMI continues and also allows for FMI-based simulators even outside of traditional CAE tools. Common platforms for FMI-based simulators include both open source and commercial offerings in a range of environments including Python (PyFMI) and MATLAB/Simulink (FMI Toolbox for MATLAB/Simulink).

Another key aspect for the efficient utilization of deployed models is the ease with which different engineering analyses can be created and executed. modeFRONTIER (ESTECO SpA, 2015) is a process integration and design optimization tool widely used in industry. The process integration platform allows multiple third party CAE tools to be coupled together to create an automated chain. With state of the art analyses capabilities and algorithms for multi-objective and multi-disciplinary optimization, robust design, sensitivity, and statistical engineering methods, modeFRONTIER offers sophisticated features to automate the design simulation process and facilitate analytic decision making. The software's advanced post-processing modules include sophisticated data

visualization and statistical tools to facilitate understanding and gain deep insights into study results.

This paper outlines a toolchain for automated deployment of models as FMUs as simulators in Microsoft Excel. The automation relies on a set of annotations in the FMU, and these annotations are fully described. By including the annotations in the Modelica code such that they are present in the generated FMUs, an automated, streamlined path from a Modelica model to a simulator in Excel is demonstrated. Integration of the automated simulators in Excel with modeFRONTIER brings a powerful suite of analysis and optimization capabilities to the simulator toolchain. Following a description of the toolchain and automation enablers, several different examples demonstrate the entire toolchain from Modelica model to deployed simulator in Excel using FMI Add-in for Excel and integrated with modeFRONTIER. These applications also highlight a range of different analysis and optimization capabilities provided by modeFRONTIER, including parameter estimation, multi-objective optimization, and robust design.

2 Toolchain Overview

This section provides a description of the entire automation toolchain. The toolchain supports any FMU annotated as described. This section describes the annotation requirements and demonstrates the inclusion of the annotations in the Modelica code to provide an automated path from Modelica model to deployed simulator in Excel with FMI Add-in for Excel (Modelon AB, 2015) and additional integration with modeFRONTIER for analysis and optimization. The entire workflow is shown in Figure 1 and described in detail in the following sections.

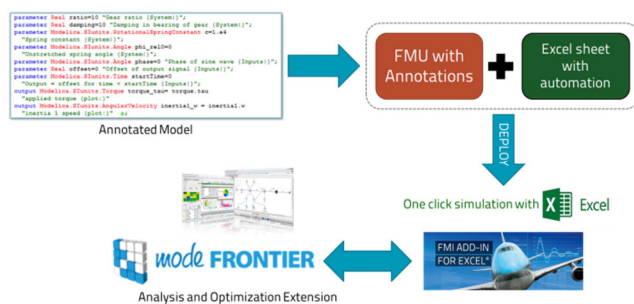


Figure 1. Workflow overview

2.1 Annotations

The toolchain automation is based on a set of annotations to identify parameters and outputs in the FMU for use in the simulator and subsequent analyses. These annotations can be provided in the Modelica code to provide a direct path for automated deployment of Modelica models as FMI-based simulators in Excel.

To identify relevant variables for the automation, the approach is to add a special substring to the variable descriptions per the markup specification in the XenGen package from Xogeny (2015). The general syntax for the markup syntax is shown below:

```
"Description
{[GroupName][Style:][LabelString]}"
```

Figure 2 shows sample annotations as implemented in Modelica code (as described in the markup specification, items in [] are optional) to identify an output variable and also a parameter. The overall steps are as follows for Modelica models:

- Annotate Modelica model to identify parameters and outputs per markup syntax
- Create FMU from Modelica model (if required, ensure export license usage)

When the Modelica code is annotated, the variable description flows directly to the FMU and is available in the variable description XML file. Thus, there is a direct path to support downstream automation that is implemented and maintained directly in the source before FMU generation. Alternatively, the FMU XML could be edited to add the annotations in cases where the original code is not accessible for markup (or even when the FMU generator is not Modelica-based). The downstream processes in the toolchain leverage only the FMU with annotations.

```
Real V(start=V0) "number of free virions {plot:}";
parameter Real lambda=80 "cell creation rate {Basic}";
```

Figure 2. Sample annotations in Modelica code

2.2 Automated Simulator in Excel

FMI Add-in for Excel (Modelon AB, 2015) provides the ability to load and simulate FMUs in Microsoft Excel. The standard workflow involves choosing the parameters and outputs to be used for experimentation via the experiment sheet which is populated with the chosen variables and ready for batch simulation. Both the final values and dynamic traces are available for post-processing. FMI Add-in also provides scripting capability for controlling the tool from macros.

Leveraging the scripting capability, automation has been added to provide automated deployment of FMUs as simulators in Excel. The automation is implemented in a workbook and provides “one click” simulation capability in Excel. This capability was first introduced to provide a dynamic simulator for small modular reactors (Hale 2014). From the main page in the workbook shown in Figure 3, the user simply points to the FMU, and the automation loads the FMU and creates an experiment sheet that includes the annotated parameter and output variables (Figure 4a). On initial load of the FMU, the workbook also runs the default simulation and plots all outputs (Figure 4b). Subsequent simulations as either single runs or batch

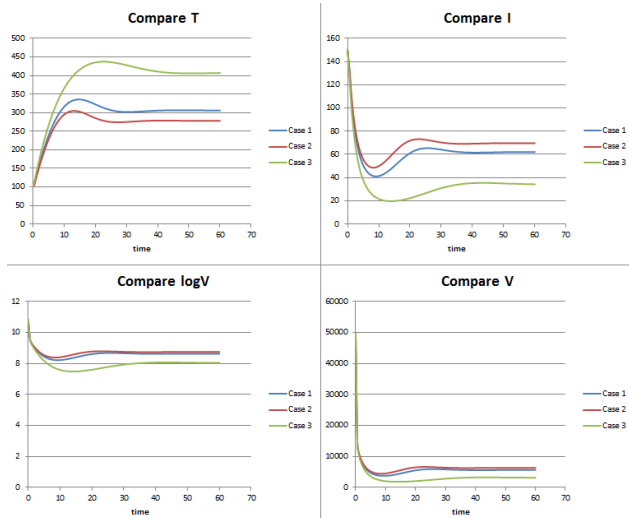
simulations are controlled by the user from the experiment sheet. Automated plotting including the ability to compare results across cases is provided in the automation worksheet. FMI Add-in for Excel offers a convenient platform in Excel for FMI-based deployment and simulation with a flexible, familiar front end for users.



Figure 3. Automated workbook main page

Model		Settings										
Sheet version	Generated by Modelon FMI Add-in for Excel version 1.3.3	Default	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9	Case 10
Model name	FMI_Demos_HIV_HV_Dynamics											
Generation tool	Dymola Version 2015 F001 (64-bit), 2014-12-15											
FMU kind	CoSimulation_StandAlone											
Number of processes	8											
Checksum	f6578c505b849c5ad5a059e61ec3754b											
Expiry date												
Start time	0											
Stop time	60											
FMU	FMI_Demos_HIV_HV_Dynamics.fmu											
Log level	Info											
Enable	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Output points	100											
Timeout	0											
Indata												
Name	Variability	Type	Unit									
lambda	parameter	Real	80									
rho	parameter	Real	0.15									
beta	parameter	Real	0.00002			2.20E-05						
delta	parameter	Real	0.55			1.50E-05						
n	parameter	Real	900									
c	parameter	Real	5.5									
T0	parameter	Real	100									
W0	parameter	Real	30000									
dV_0	parameter	Real	-200750									
Outdata												
Name	Variability	Type	Unit									
T	continuous	Real		305.829	278.039	406.843						
I	continuous	Real		62.0801	69.616	34.2993						
V	continuous	Real		5581.11	6258.58	3084.23						
logV	continuous	Real		8.62734	8.74171	8.03406						
Message				OK	OK	OK	Disablec	Disablec	Disablec	Disablec	Disablec	Disablec

(a) Experiment sheet for batch simulation



(b) Plotting sheet comparing cases

Figure 4. Sample experiment and plot sheets from workbook automation

2.3 Integration with modeFRONTIER

As an analysis and optimization tool, modeFRONTIER integrates with many different CAE tools and modeling formalisms (lumped parameter, CFD, FEA, etc.). Currently modeFRONTIER does not include native FMU capability. Since modeFRONTIER includes a widely-used Excel interface, adding FMU simulation capability easily via FMI Add-in for Excel is a natural extension. Integration between FMI Add-in for Excel and modeFRONTIER leverages the existing Excel interface in modeFRONTIER and does not require any customization thereby maintaining a consistent workflow and user interface. With existing capabilities to interact with Excel sheets, modeFRONTIER leverages the deployed simulator in Excel via FMI Add-in for Excel to set parameters, simulate the FMUs, and extract data from the experiment sheet. The experiment sheet in FMI Add-in for Excel is treated no differently than any other Excel sheet with which modeFRONTIER can interact. A macro to trigger the simulation is provided in the automated worksheet. Both modeFRONTIER and FMI Add-in for Excel can parallelize the simulation runs across available machine cores for maximum utilization of computing resources. Figure 5 shows the Excel node configuration for a sample deployed simulator in FMI Add-in for Excel. The node configuration provides modeFRONTIER with the cell locations (or named ranges) for inputs and outputs along with the workbook location and macro to trigger the simulation. Multiple Excel nodes with FMUs can be coupled in a workflow (note that this coupling does not provide transient coupling between FMUs, but this coupling can be enabled in other FMI co-simulation master tools from which a single FMU can be created for use in an Excel node).

This off-the-shelf integration between FMI Add-in for Excel and modeFRONTIER provides FMU simulation capabilities to support a wide variety and rapidly growing list of third party tools with FMI support.

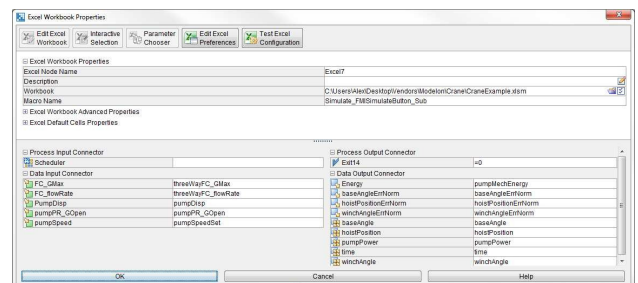


Figure 5. Excel node configuration in modeFRONTIER as applied to a deployed simulator in FMI Add-in for Excel

3 Application Examples

This section of the paper describes several different example models to illustrate the integration and analysis capabilities of modeFRONTIER and Modelica models deployed as FMU-based simulators with FMI Add-in for Excel. Co-simulation FMUs for all models are created using Dymola (Dassault Systemes, 2015). A brief overview of each model will be provided along with a description of the analysis problem, formulation of the workflow in modeFRONTIER, and key results.

3.1 HIV Virus Dynamics

As a simple first example of a dynamic system, a model of the dynamics of the HIV virus in human blood (Soetart and Petzoldt, 2010) has been implemented in Modelica. The model is similar to the standard predator-prey model for the number of uninfected (T) and infected (I) cells and the number of free virions (V). The schematic of the pathways is shown in Figure 6. The model is implemented directly in Modelica as it consists of three differential equations and has a number of parameters which are typically estimated based on clinical data from patients.

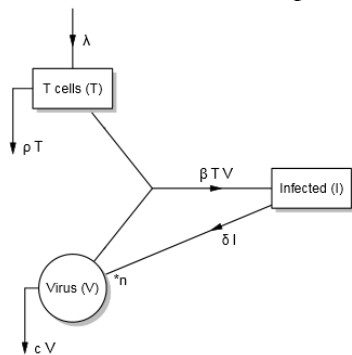


Figure 6. HIV dynamics model showing creation, destruction, and transition paths between cells (Soetart and Petzoldt, 2010)

modeFRONTIER uses a graphical workflow to set up and execute analysis and optimization tasks. These workflows can involve a single simulation node or multiple simulation nodes connected together to construct more complex, multidisciplinary tasks involving multiple tools and modeling formalisms. Figure 7 shows the workflow used to execute the HIV dynamics model correlation application. The Excel node with the FMU simulation is in the middle and labeled “FMIE”. The inputs/parameters are shown at the top of the diagram. The workflow starts by executing a Design of Experiments (DOE) for initial data. The outputs are shown at the bottom of the diagram along with any constraints or post-processing calculations for the algorithms. This application uses ESTECO’s proprietary FAST (Montrone, 2014) strategy to estimate the model parameters to best fit data. The data fitting is applied to the transient T

curve. The FAST strategy applies an algorithm to response surface models (RSM) to accelerate the fitting procedure. In this case, the SIMPLEX (Poles, 2003) algorithm was applied.

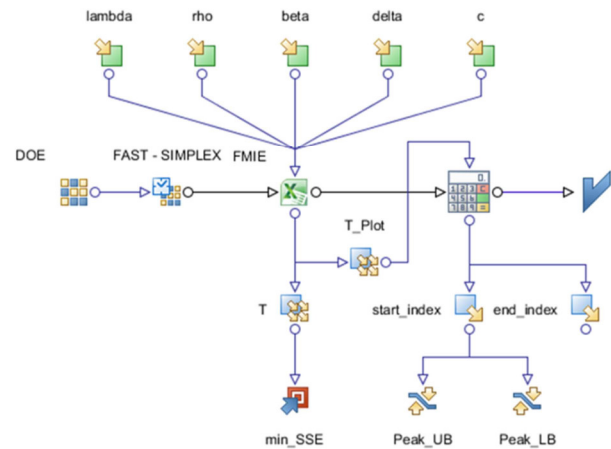


Figure 7. modeFRONTIER workflow for model correlation graphically representing the problem statement with inputs, outputs, target objective, constraints, and process flow

The optimization convergence history and dynamic T curves can be seen in Figure 8 and Figure 9, respectively. The initial T curve and the curve after fitting to the target data are shown in Figure 10.

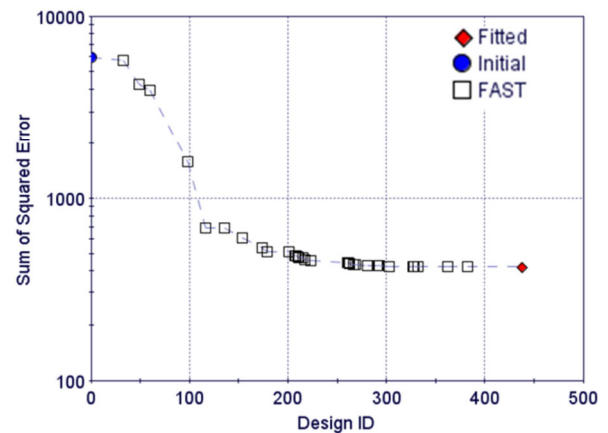


Figure 8. Optimization history showing only improved designs

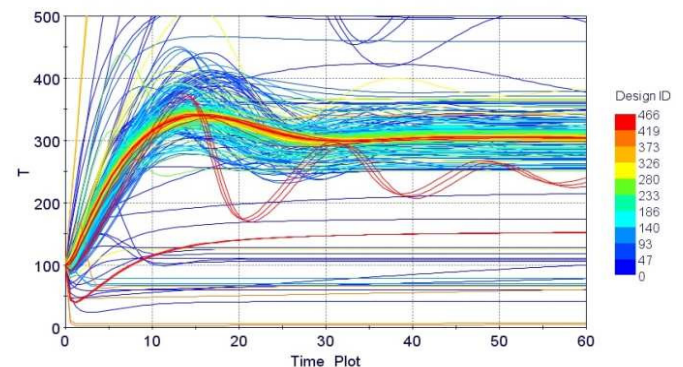


Figure 9. History as the T curve converges to the target data

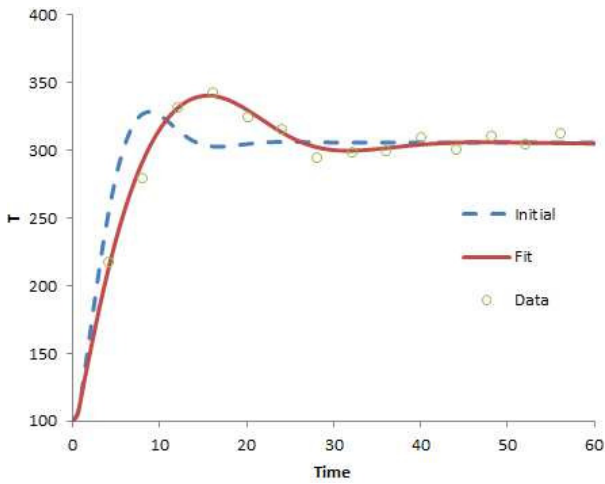


Figure 10. Initial T curve and fit curve to target data

3.2 Hydraulic Crane

Figure 11 shows a model of a crane system with a hydraulic system with motors for the movement of the crane and load. The base motor, winch motor, and hoist jack are all position controlled in closed loop to meet a desired trajectory for the crane and load. A screenshot of the animation of the model from Dymola is shown in Figure 12 with a trace for the movement of the load.

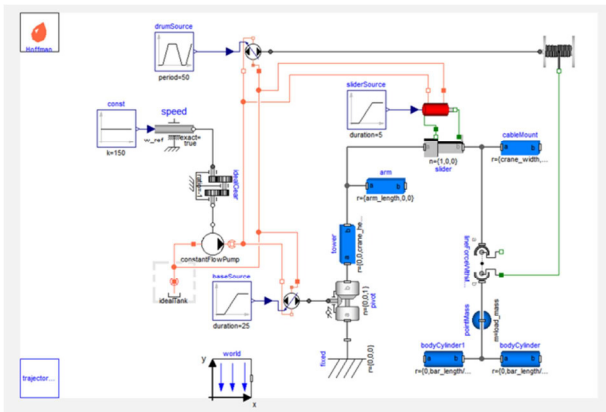


Figure 11. Crane model with hydraulic system

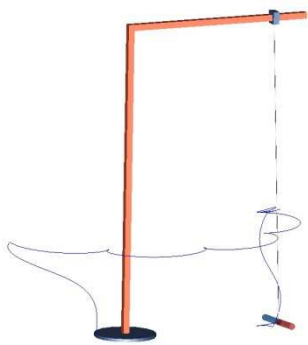


Figure 12. Crane animation

This application is formulated as a multi-objective optimization problem for modeFRONTIER via the workflow shown in Figure 13. The objective is to minimize the total tracking error and pump mechanical energy required to move the load. The total error objective function is the summation of the winch angle, base angle, and hoist position errors. Potential variables for optimization include the pump characteristics, actuator characteristics, and actuator control parameters. The input variable bounds are listed in Table 1.

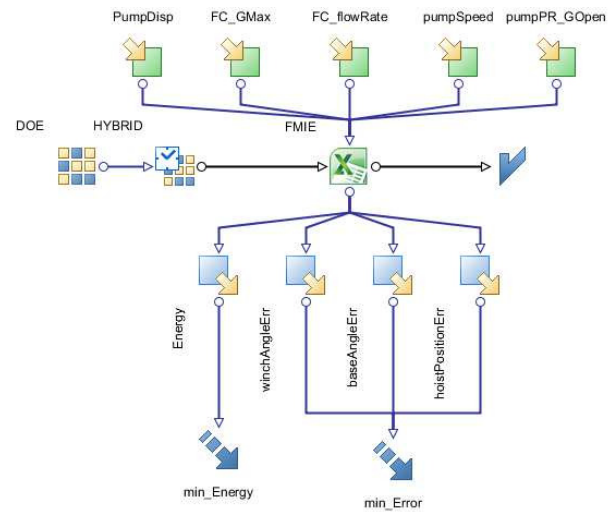


Figure 13. modeFRONTIER workflow for crane optimization showing inputs fed into Excel plugin and outputs with objective functions applied

Table 1. Input variable bounds

Name	Description	Lower Bound	Upper Bound	Units
pumpDisp	Pump Displacement	1.00E-04	0.005	m ³
FC_GMax	threeWay FC_Gmax	1.00E-11	1.00E+08	m ³ /(s.Pa)
FC_flowRate	threeWay FC_flowRate	0.001	0.05	m ³ /s
pumpSpeed	Pump SpeedSet	50	300	rad/s
PR_GOpen	Pump PR_Gopen	1.00E-12	1.00E-05	m ³ /(s.Pa)

ESTECO’s proprietary HYBRID (Turco, 2011) algorithm was used along with a Multi-Objective Genetic Algorithm (MOGA-II) (Poles, 2003). The HYBRID algorithm combines the global exploration capabilities of a genetic algorithm with a gradient based method. Using both HYBRID and MOGA-II algorithms gave improved coverage of the pareto front as shown in Figure 14.

A uniformly distributed Latin Hypercube design of experiments of 10 points was used as the starting population for both algorithms. HYBRID generated a total of 700 designs; MOGA-II generated a total of 1300 designs. The pareto results in Figure 14 illustrate

the trade-off between the error and pump energy objectives.

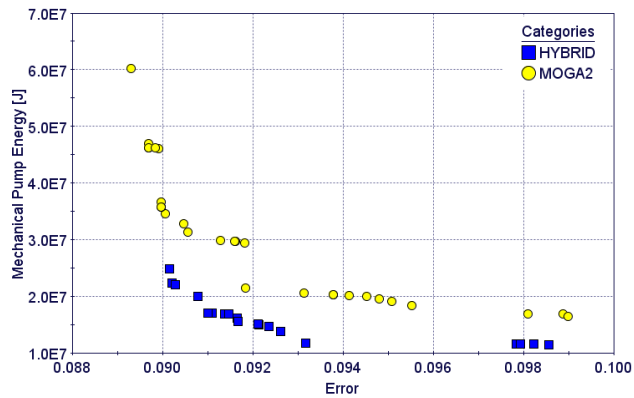


Figure 14. Pareto results from two separate optimizations using HYBRID and MOGA-II algorithms

Further analysis was done using modeFRONTIER’s Multivariate tool (Stefano, 2009) to detect clusters in the pareto data. Hierarchical clustering was performed on the pareto points combining the results from both algorithms. Three clusters were identified and are represented as the three bands shown in Figure 15. The width of the band represents a 90% confidence interval. A different local response of the system is expected for each cluster. Rather than choosing a single design point from among the pareto set, a cluster can be chosen. This approach narrows down the number of candidate designs while also offering variability within the confidence interval region. In this case, we have three clusters where the energy and error objectives are distinctly different. Each cluster’s corresponding configuration can be seen in Figure 15.

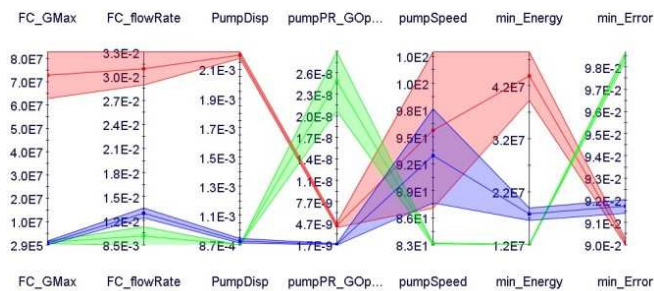


Figure 15. Clustering results on the pareto points showing three distinct pareto solutions (band width represents 90% confidence interval)

3.3 Heat Exchanger Performance with Blockage

Heat exchanger performance under non-uniform boundary conditions is a critical analysis need for vehicle thermal management (Batteh *et al.*, 2014). Blockage due to heat exchanger stacking, geometric interference with the vehicle body, or even fouling can drastically affect heat exchanger performance.

Figure 16 shows a heat exchanger test bench implemented with Modelon Heat Exchanger Library (Modelon AB, 2015). Non-uniform air side boundary conditions are provided across the face of the heat exchanger model using the Modelon DataAccess package. DataAccess provides XML reading capability and preserves dynamic file access even with the model exported as an FMU. Figure 17 shows the simple blockage pattern simulated for the cooler where the first 25% of the cooler is completely blocked.

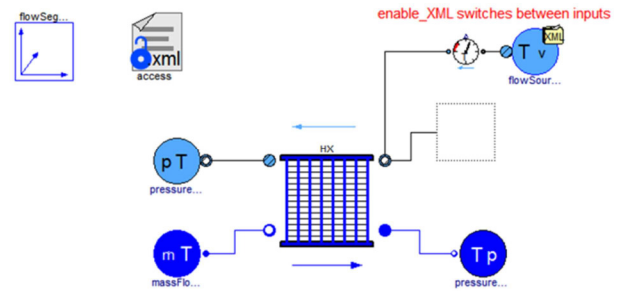


Figure 16. Heat exchanger test bench with non-uniform boundary conditions via XML

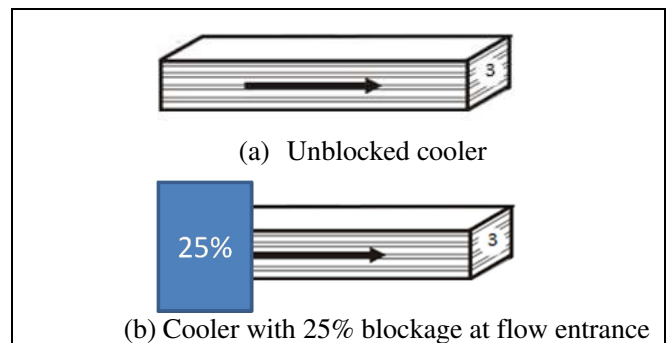


Figure 17. Heat exchanger blockage pattern

The goal of this application is to identify the velocity scale factor required for the blocked cooler such that the heat transfer performance matches that of the unblocked cooler under the same boundary conditions. For this case, the desired heat flow rate $Q_{desired}$ is 38.31kW. In addition, a robustness constraint is applied to ensure that the heat transfer does not drop by more than 1% for a 5% reduction in airflow. The workflow for modeFRONTIER is shown in Figure 18. This problem is executed in modeFRONTIER as a robust design optimization (RDO) using the SIMPLEX algorithm. A schematic showing the heat transfer distribution as a function of airflow distribution is shown in Figure 19 for an unfeasible and feasible solution based on the problem definition to illustrate the approach used for robust design optimization.

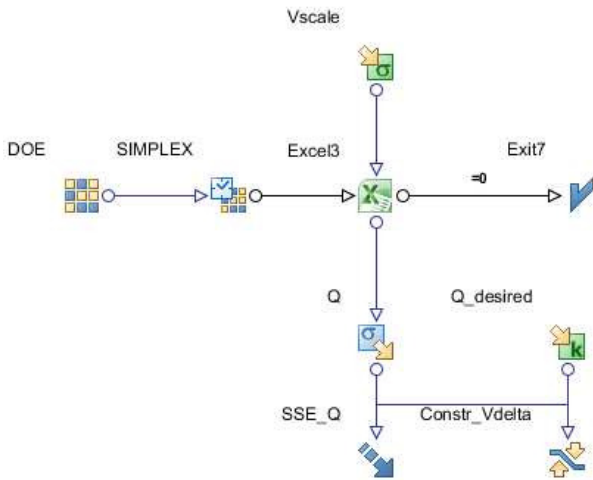


Figure 18. modeFRONTIER workflow for heat exchanger robust design application showing the stochastic input V_{scale} and reliability objective SSE_Q with constraint $Constr_Vdelta$

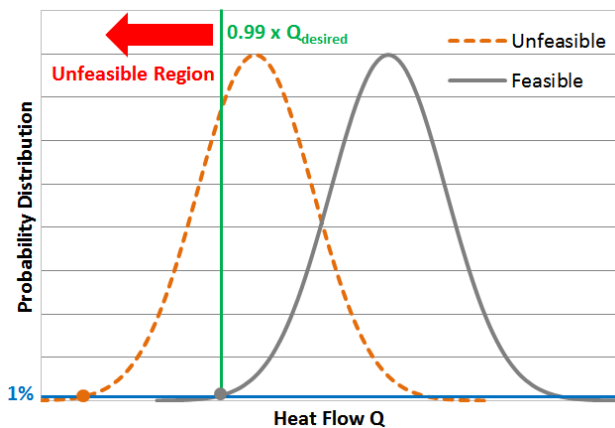


Figure 19. Schematic of feasible and unfeasible run from robust design optimization

The results of the robust design optimization are shown in Figure 20. Note the series of runs around the V_{scale} value of 1.88 which provides the required heat transfer but falls just at the constraint boundary. The value of V_{scale} that meets both the heat transfer requirement and the robustness constraint is slightly higher. While a relatively straight forward application with only a single design variable, this problem becomes significantly more complex with several design constraints and multiple design variables.

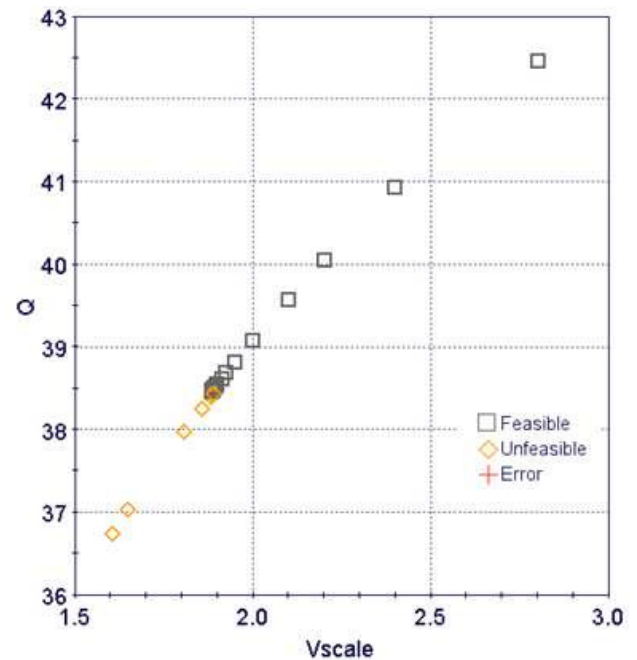


Figure 20. Convergence of the robust design optimization runs for the heat exchanger blocking problem

3.4 Hybrid Vehicle Electric Range

Vehicle range is a key metric for hybrid vehicles with electric-only mode. In addition to key vehicle parameters which affect the loads and losses, critical battery parameters strongly affect vehicle range. Battery performance is affected by both battery temperature and battery age. As batteries age, their capacity decreases while the internal resistance increases and leads to larger heat losses. Particular battery formulations typically have an optimal temperature operating range and performance degrades when the temperature strays above or below the range.

Figure 21 shows a series hybrid truck model implemented with the Modelon Vehicle Powertrain package which uses the PowerTrain Library (DLR, 2015). The battery model is a table-based model that provides open circuit voltage and internal resistance as a function of current and battery State-Of-Charge (SOC). However, the model has been enhanced to include temperature scaling in the battery tables and an approximate aging model that increases internal resistance and decreases capacity based on an aging factor.

Since actual vehicle populations in the field will have some sort of aging distribution based on usage, the resulting vehicle electric range for the fleet will be a distribution. It is important to understand the impact of aging and temperature on vehicle electric range and also analytically determine fleet population distributions.

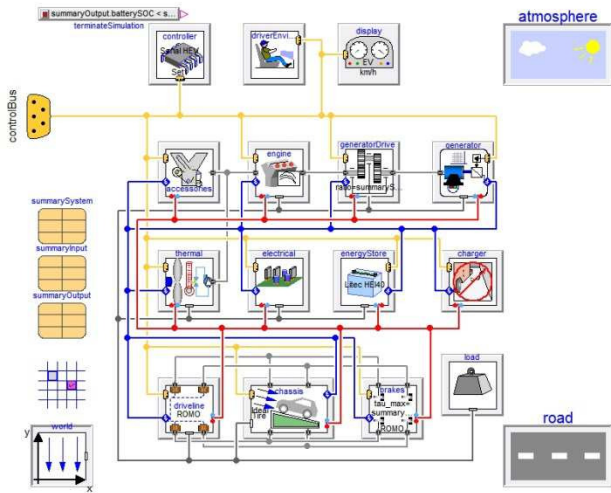


Figure 21. Series hybrid truck model

One approach to analytically determine populations is to assume a distribution and then run Monte Carlo simulations to estimate the fleet output distribution. modeFRONTIER can certainly perform Monte Carlo analyses. However, for this simple example, it is possible to run the simulations over the battery aging factor and then simply construct the distributions offline, thereby saving computational resources. The modeFRONTIER workflow for the electric vehicle range mapping is shown in Figure 22. For a given battery sink temperature, Latin Hypercube sampling is used to span the battery age factor. The simulations are run starting with a battery SOC=1 until the SOC is depleted via repeated execution of the New European Driving Cycle (NEDC) cycles. Due to numerical effects around zero SOC, the simulation is terminated at an epsilon SOC (0.05). In real practice, to prevent damage, a battery is never over charged or discharged. When studying relative effects (e.g. battery age on vehicle range), the absolute minimum SOC is not critical.

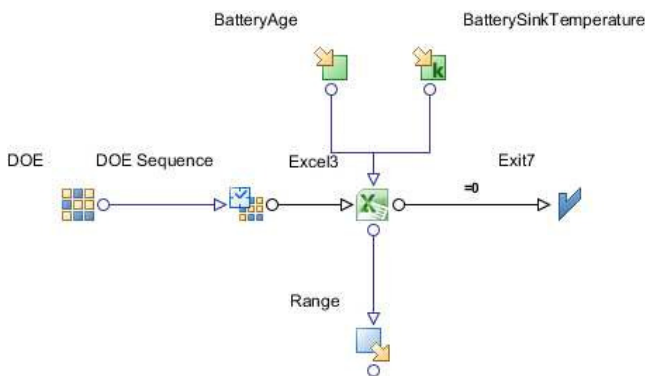
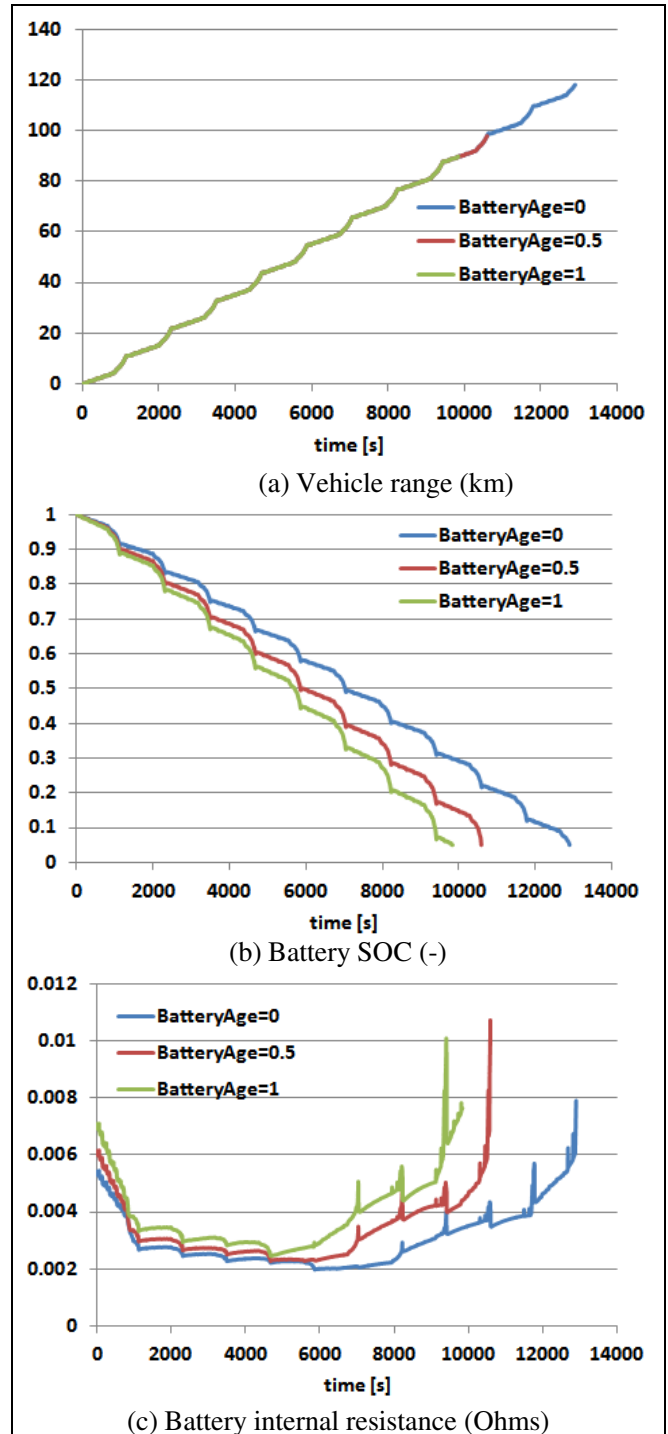


Figure 22. modeFRONTIER workflow for electric vehicle range application

Vehicle simulation results are shown in Figure 23 for a battery with no aging (BatteryAge=0), mid aging (BatteryAge=0.5), and extended aging (BatteryAge=1). As battery aging increases, vehicle range decreases substantially and battery temperatures increase due to higher internal resistance and the passive cooling strategy employed in this model. The modeFRONTIER runs showing vehicle Range over the full BatteryAge distribution are shown in Figure 24.



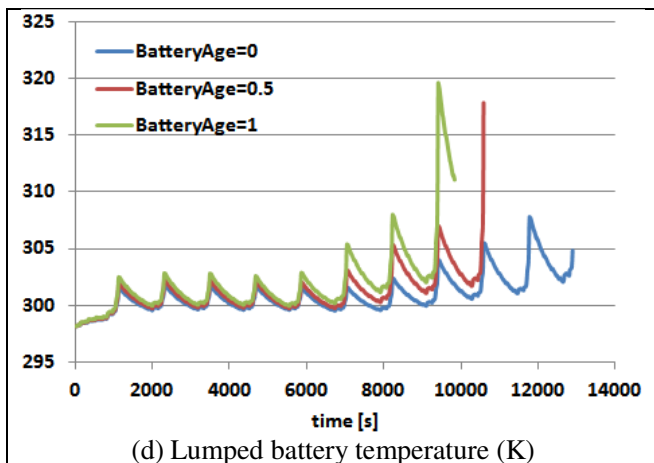


Figure 23. Simulation results for BatteryAge=0 (blue), BatteryAge=0.5 (red), and BatteryAge=1 (green)

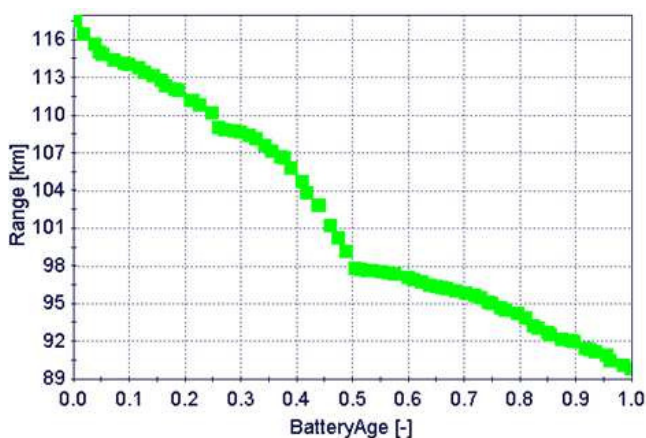


Figure 24. modeFRONTIER runs showing Range over BatteryAge distribution

With the results from modeFRONTIER over the entire age range, any arbitrary vehicle age distribution can be assumed and the fleet range distribution calculated. Figure 25 shows the calculated distribution for a normal BatteryAge distribution with a mean=0.5 and standard deviation=0.05. Similar calculations could be done over a range of different drive cycles, battery ages, and temperatures to estimate more complex fleet populations.

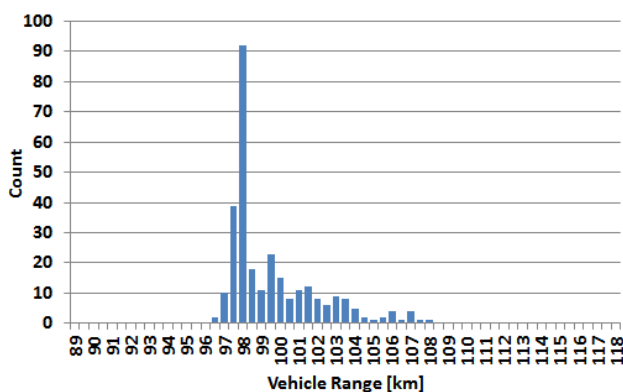


Figure 25. Fleet population assuming normal distribution with BatteryAge=0.5 and standard deviation=0.05

4 Conclusions

This paper demonstrates a method for automated deployment of models as FMU-based simulators in Microsoft Excel using FMI Add-in for Excel. A method for annotating the Modelica code using the XenGen markup syntax supports the automation to provide a streamlined path from a Modelica model to a deployed simulator in Excel. Integration of the automated simulators in Excel with modeFRONTIER brings a powerful suite of analysis and optimization capabilities to the simulator toolchain. Several different examples demonstrate the entire toolchain from Modelica model to deployed simulator in Excel using FMI Add-in for Excel and integrated with modeFRONTIER. These applications also highlight a range of different analysis and optimization capabilities provided by modeFRONTIER, including parameter estimation, multi-objective optimization, and robust design. This toolchain can be applied to any FMU and streamlined with automation enabled by the supporting annotations.

References

- John Batteh, Jesse Gohl, and Chandrasekar Sureshkumar. Integrated Vehicle Thermal Management in Modelica: Overview and Applications, *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, pp. 409-418, March 2014. doi: 10.3384/ecp14096409.
- Dassault Systemes. Dymola 2015 FD01, 2015.
- DLR, PowerTrain Library, v.2.3.0, 2015.
- ESTECO SpA. modeFRONTIER 2014 Update 1, 2015. <http://www.esteco.com/>
- Richard Hale, Sacit Cetiner, David Fugate, Lou Qualls, John Batteh, and Michael Tiller. Dynamic Modeling of Small Modular Nuclear Reactors using MoDSim, *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, pp. 989-998, March 2014. doi: 10.3384/ecp14096989.
- Modelon AB, FMI Add-in for Excel, v1.3.5, 2015. <http://www.modelon.com/products/fmi-add-in-for-excel/>.
- Modelon AB, Heat Exchanger Library, v1.2, 2015. <http://www.modelon.com/products/modelica-libraries/heat-exchanger-library/>.
- T. Montrone, A. Turco, R. Enrico. FAST Optimizers: General Description. *ESTECO Technical Report 2014-001*, Trieste, Italy, 2014.
- S. Poles. The SIMPLEX Method. *ESTECO Technical Report 2003-005*, Trieste, Italy, 2003.
- S. Poles. MOGA-II an Improved Multi-Objective Genetic Algorithm. *ESTECO Technical Report 2003-006*, Trieste, Italy, 2003.
- Karline Soetaert and Thomas Petzoldt. Inverse Modelling, Sensitivity and Monte Carlo Analysis in R using Package FME. *Journal of Statistical Software*, 33(3), pp. 1-28, February 2010.

- D. Stefano. Multivariate Analysis algorithms in modeFRONTIER v4. *ESTECO Technical Report 2009-001*, Trieste, Italy, 2009.
- A. Turco. Hybrid – description. *ESTECO Technical Report 2011-003*, Trieste, Italy, 2011.
- Xogeny, XenGen package, 2015.
<https://github.com/xogeny/XenGen>

An Open-Source Graphical Composite Modeling Editor and Simulation Tool Based on FMI and TLM Co-Simulation

Alachew Mengist¹, Adeel Asghar¹, Adrian Pop¹, Peter Fritzson¹, Willi Braun², Alexander Siemers³,
Dag Fritzson³

¹PELAB – Programming Environment Lab, Dept. Computer Science, Linköping University, Sweden,
{alachew.mengist, adeel.asghar, adrian.pop, peter.fritzson}@liu.se

²Dept. Mathematics and Engineering, University of Applied Sciences, Germany, willi.braun@fh-bielefeld.de

³SKF, Göteborg, Sweden, {alexander.siemers, dag.fritzson}@skf.com

Abstract

A common situation in industry is that a system model (here a composite model) is composed of several sub-models which may have been developed using different tools. FMI is one important technology for exporting/importing models between tools and/or connecting them via co-simulation. TLM based modeling and co-simulation is another important technique for modeling, connecting, and simulation of especially mechanical systems, which is simple, numerically stable, and efficient. A number of tool-specific simulation models, such as Modelica models, SimuLink models, Adams models, BEAST models, etc., have successfully been connected and simulated using TLM based co-simulation. However, previously there was no general open source tool for creation, graphic editing, and simulation of composite models connected via FMI or TLM based co-simulation. In this paper we present a graphical composite model editor based on OpenModelica which is integrated with the OpenModelica and the SKF TLM co-simulation frameworks to support both FMI and TLM based composite model editing and simulation. The editor supports creating, viewing and editing a composite model both in textual and graphical representation. The system supports simulation of composite models consisting of sub-models created using different tools.

Keywords: Graphic Editing, Composite Modeling, Modelica, FMI, TLM, XML, Simulation, Co-Simulation

1 Introduction

Industrial products often consist of many components which have been developed by different suppliers using different modeling and simulation tools. Modeling and simulation support is needed in order to integrate all the parts of a complex product model. FMI (Blochwitz et al, 2011), (FMI-Standard.org, 2014), both model exchange and co-simulation, is one such important technology. TLM (Transmission Line Modeling) based co-simulation (Siemers et al, 2005), is another important technology, which is simple, numerically stable, and efficient.

This has successfully been demonstrated by integrating and connecting several different simulation models, especially for mechanical applications. Such an integrated model consisting of several model parts is here called a *composite model* since it is composed of several sub-models. Another name used for such a model is *meta-model*, since it is a model of models. In earlier work (Siemers et al, 2005), (Siemers and Fritzson, 2006) Modelica (Fritzson, 2014) with its object oriented modeling capabilities and its standardized graphical notations has demonstrated the possibilities for meta-modeling/composite modeling of mechanical systems using TLM.

The availability of a general XML-based composite modeling language (Siemers et al, 2005) is an important aspect of our FMI and TLM based modeling and co-simulation framework. However, modelers developing composite models are likely to take advantage of the additional availability of tools that assist them with respect to the composite modeling process (i.e., the process of creating and/or editing a composite model, here represented and stored as XML).

We introduce a graphical composite model editor which is an extension and specialization of the OpenModelica connection editor OMEdit (Asghar et al, 2010). In the context of this work a composite model is composed of several sub-models including the interconnections between these sub-models. The graphical editor presented in this paper enables users to create a composite model in XML. It is also integrated with the OpenModelica FMI-based model-exchange and co-simulation and the SKF TLM-based co-simulation framework.

This paper is structured as follows. In Section 2 a brief background of OpenModelica is given along with an overview of the SKF TLM based co-simulation framework. Section 3 defines the different modes of operation. The composite model XML schema is explained in Section 4. In Section 5 the composite model editor and an overview of its interaction with the other system components are discussed. An industrial application is shown in Section 6. Conclusions and future work are presented in Section 7.

2 Background

2.1 OpenModelica

OpenModelica is an open source Modelica-based platform for modeling, simulation, and optimization. The OpenModelica connection editor OMEdit is a graphical Modelica model editing tool. It supports model creation, textual and graphical model editing including connections drawing, simulation and plotting. This editor has been extended to become a composite model editor as described in this paper.

2.2 FMI-Based Model Exchange and Co-Simulation Framework in OpenModelica

OpenModelica currently supports FMI 1.0 and FMI 2.0 for model exchange and most of FMI 2.0 for co-simulation. Other tools can access this functionality e.g., by dynamically linking OMC or by invoking it using a message-passing interface.

The OpenModelica graphic editor and simulator supports connection of several FMUs into a composite model and simulating this model.

Simulation of composite models with algebraic loops involving FMUs is also supported.

2.3 TLM (Transmission Line Modeling)

The TLM (Transmission Line Modeling) technique is based on the fact that propagation of signals takes time in many physical systems, e.g., propagation of pressure waves in hydraulic systems, sound wave propagation, and force propagation in mechanical systems. Thus there is a certain physical communication delay between different parts of the system which can be used to partially de-couple these parts and allow them to be independently simulated and coupled in a numerically stable way via co-simulation techniques.

The TLM approach has been known since a long time (Auslander, 1968), (Burton et al, 1993), (Johns and O'Brien 1980). A general presentation of the theory and methods of TLM-based simulation is given in (Krus et al, 1990).

The HOPSAN (HOPSAN, 1985) software is one of the first general TLM implementations with its own graphical modeling language. A newer version, HOPSAN-NG, (Axin et al, 2010), has recently been developed.

Moreover, a TLM implementation for the Modelica language has recently been developed as part of OpenModelica (Chapter 7, Sjölund, 2015).

2.4 TLM-Based Co-Simulation Framework

As mentioned, a general framework for composite model based co-simulation has previously been designed and implemented (Siemers et al, 2005). The design goals for the simulation part of that framework were portability, simplicity to incorporate additional simulation tools, and computational efficiency. It is

also the framework used for TLM-based composite model co-simulation described in this paper.

The TLM composite model co-simulation is primarily handled by the central simulation engine of the framework called the TLM simulation manager. It is a stand-alone program that reads an XML definition of the coupled simulation as defined in (Siemers et al, 2005). It then starts external model simulations and provides the communication bridge between the running simulations using the TLM (Nakhimovski, 2006) method.

The external models only communicate with the TLM simulation manager which acts as a broker and performs communication and marshalling of information between the external models. The simulation manager sees every external model as a black box having one or more external interfaces. The information is then communicated between the external interfaces belonging to the different external models. Additionally the simulation manager opens a network port for monitoring all communicated data.

TLM simulation monitor is another stand-alone program that connects to the TLM simulation manager via the network port. The TLM simulation manager sends the co-simulation status and progress to the TLM simulation monitor via TCP/IP. The simulation monitor receives the data and writes it to an XML file.

3 Modes of Operation

3.1 Textual Format

The user defines the list of sub-models with their corresponding connections as an XML file according to the composite model specification described in Section 4. The user should be able to easily save the file, load a new one, or edit the textual version even while using the GUI. The simulator reads the composite model XML file and performs the simulation.

3.2 Graphical User Interface Format

The user can define the list of sub-models and their connections using the GUI which allows them to drag and drop the sub-models to the editor and make connections between them. The GUI automatically generates the composite model code which is stored in an XML format described in Section 4.

4 Composite Model XML Schema

The composite model XML-Schema for validating the co-simulation composite model is designed according to its specification described in (Siemers et al, 2005). The following is a sample composite model XML representation:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Model Name="Pendulum">
```



```

<SubModels>
  <SubModel Name="shaft1"
  StartCommand="StartTLMOpenModelica"
  ExactStep="0" ModelFile="shaft1.mo"
  Position="0.0,0.0,0.0"
  Angle321="0.0,0.0,0.0">
    <InterfacePoint Name="tlm"
    Position="0.0,0.0,0.5"
    Angle321="0.0,0.0,0.0"/>
  </SubModel>
  <SubModel Name="shaft2"
  StartCommand="StartTLMOpenModelica"
  ExactStep="0" ModelFile="shaft2.mo"
  Position="0.0,0.0,0.5"
  Angle321="0.0,0.0,0.0">
    <InterfacePoint Name="tlm"
    Position="0.0,0.0,0.0"
    Angle321="0.0,0.0,0.0"/>
  </SubModel>
</SubModels>
<Connections>
  <Connection From="shaft1.tlm"
  To="shaft2.tlm" Delay="1e-4" Zf="1e4"
  Zfr="1e2" alpha="0.2"/>
</Connections>
<SimulationParams StartTime="0"
  StopTime="5"/>
</Model>

```

In order to use graphical notations in the composite model editor, the composite model XML file needs to describe annotations for each sub-model and connections between them. We propose to extend the composite model specification by including the Annotation element in the SubModel and Connection elements.

```

<Annotation Origin="{-50,54}" Extent="{-
  10,-10,10,10}" Rotation="0"
  Visible="true"/>

```

The contents of our composite model XML root element, namely Model is depicted in Figure 1. Inside the root element there can be a list of connected SubModels and TLM Connections. SimulationParams element is also inside the root element. It has an attribute Name representing the name of the composite model.

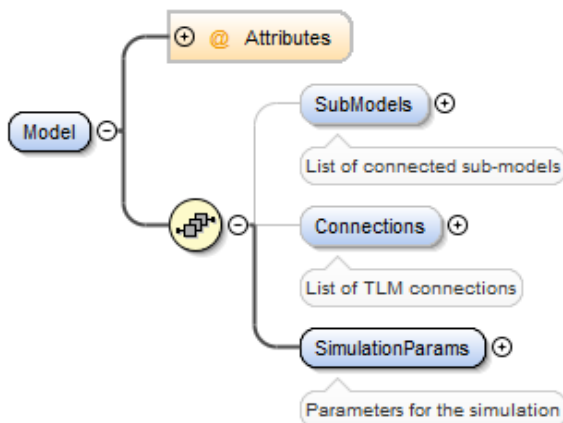


Figure 1. The Model (root) element of the Composite Model Schema.

The SimulationParams element specify the start time and end time for the co-simulation

The SubModel element, presented in Figure 2, represents the simulation model component that participates in the co-simulation. The required attribute for a SubModel are Name of the sub-model, ModelFile (file name of the submodel) and StartCommand (the start method command to participate in the co-simulation). Each SubModel also contains a list of interface points. InterfacePoint elements are used to specify the TLM interfaces of each simulation component (sub-model).

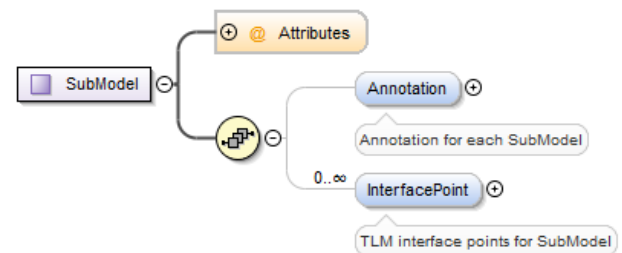


Figure 2. The SubModel element from the Composite Model Schema.

The Connection element of the composite model xml schema is shown in Figure 3.

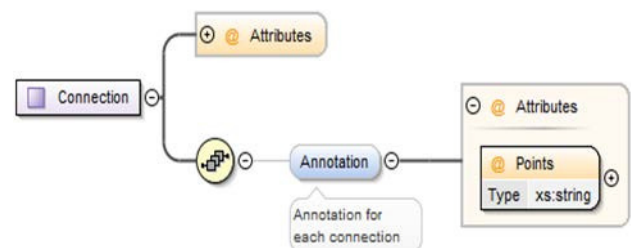


Figure 3. The Connection element from the Composite Model Schema.

The Connection element defines connections between two connected interface points, that is, a connection between two TLM interfaces. Its attributes From and To define which interface of which sub-models are connected. Other attributes of the Connection element specify the delay and maximum step size.

5 Composite Model Graphical Editor

One of the primary contributions of this effort is our focus on interoperability in modeling and simulation. Our effort leverage OpenModelica for graphical composite model editing as well as FMI support and SKF's co-simulation framework for TLM Based co-simulation.

As mentioned, the implementation of this graphical composite model editor is an extension of OMEdit (Asghar et al, 2010) which is implemented in C++ using the Qt graphical user interface library.

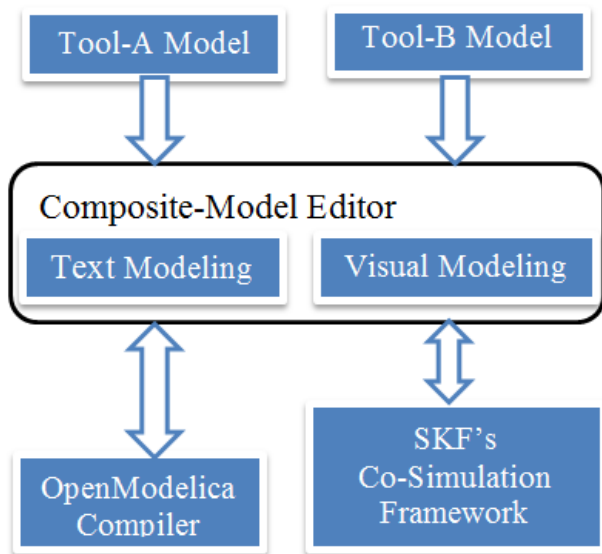


Figure 4. An overview of the interaction between the composite model (meta-model) graphic editor and the other components.

The full graphical functionality of the composite modeling process can be expressed in the following steps:

- Import and add the external models to the composite model editor,
- Specify startup methods and interfaces of the external model,
- Build the composite models by connecting the external models,
- Set the co-simulation and TLM parameters in the composite model.

An overview of the different components that the graphical composite model editor relies on is shown in Figure 4.

The graphical composite model editor communicates with the OpenModelica compiler to retrieve the interface points for the external model and SKF's co-simulation framework to run the TLM simulation manager and simulation monitor. Each tool component is described in the following subsections.

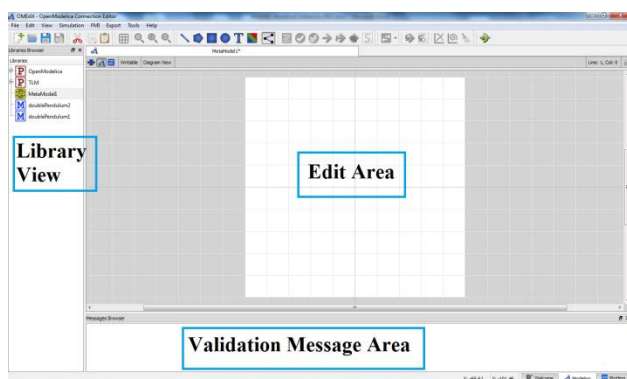


Figure 5. A screenshot of the modeling page area.

In the graphic composite model editor the modeling page area is used for visual composite modeling or text composite modeling. This allows users to create, modify, and delete sub-models. A screenshot of the modeling page area is shown in Figure 5.

5.1 Visual Modeling

Each composite model has two views: a Text view and a Diagram view. In the Diagram view, each simulation model component (sub-model) of the TLM co-simulation can be dragged and dropped from the library browser to this view, and then the sub-model will be automatically translated into a textual form by fetching the interface name for the TLM based co-simulation. The user can complete the composite model (see Figure 6) by graphically connecting components (sub-models).

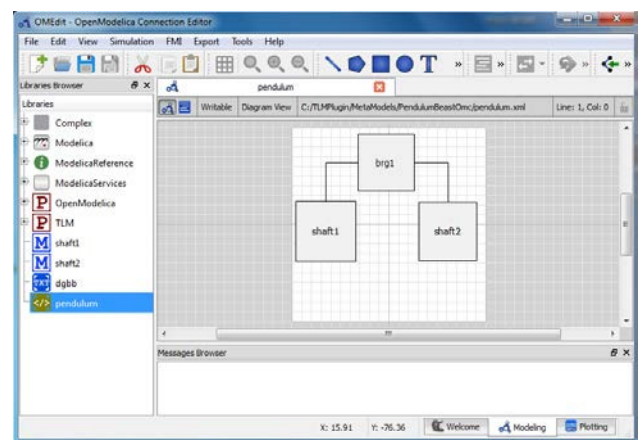


Figure 6. A screenshot of visual composite modeling after a connection has been made.

The test model (see Figure 7) is a multibody system that consists of three sub-models: Two OpenModelica Shaft sub-models (Shaft1 and Shaft2) and one SKF/BEAST bearing sub-model that together build a double pendulum. The SKF/BEAST bearing sub-model is a simplified model with only three balls to speed up the simulation.

Shaft1 is connected with a spherical joint to the world coordinate system. The end of Shaft1 is connected via a TLM interface to the outer ring of the BEAST bearing model. The inner ring of the bearing model is connected via another TLM interface to Shaft2. Together they build the double pendulum with two shafts, one spherical OpenModelica joint, and one BEAST bearing.

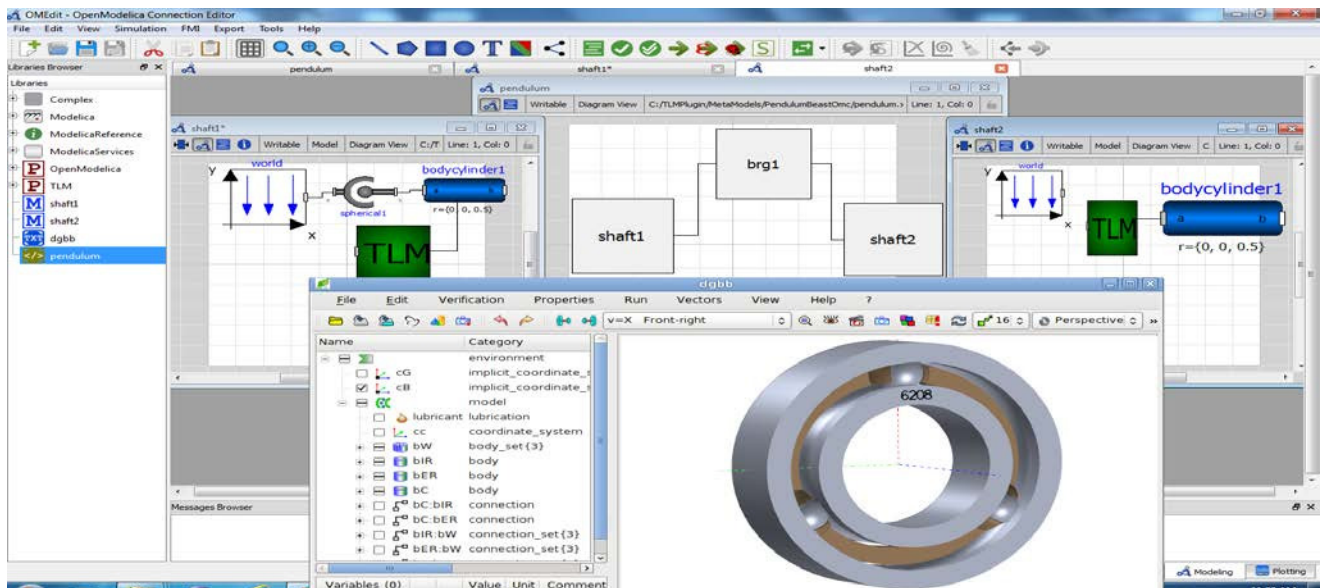


Figure 7. A screenshot of visual composite modeling of double pendulum.

5.2 Text Modeling and Viewing

The text view (see Figure 8) allows users to view the contents (sub-models, connections, and simulation parameters) of any loaded composite model. It also enables users to edit a composite model textually as part of the composite modeling construction process. To facilitate the process of textual composite modeling and to provide users with a starting point, the text view (see Figure 8) includes the composite model XML schema elements and the default simulation parameters.

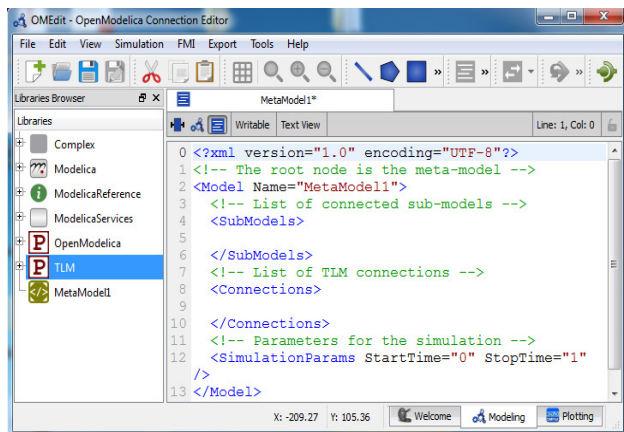


Figure 8. A screenshot of textual composite modeling.

5.3 Composite Model Validation

Since model validation is part of the composite modeling process the composite model editor (see Figure 9) supports users by validating the composite model to ensure that it follows the structure and content rules specified in the composite model schema described in Section 4. In general the composite model editor validation mechanism supports users to verify that:

- The basic structure of the elements and attributes in the composite model matches the composite model schema.
- All information required by the composite model schema is present in the composite model.
- The data conforms to the rules of the composite model schema.

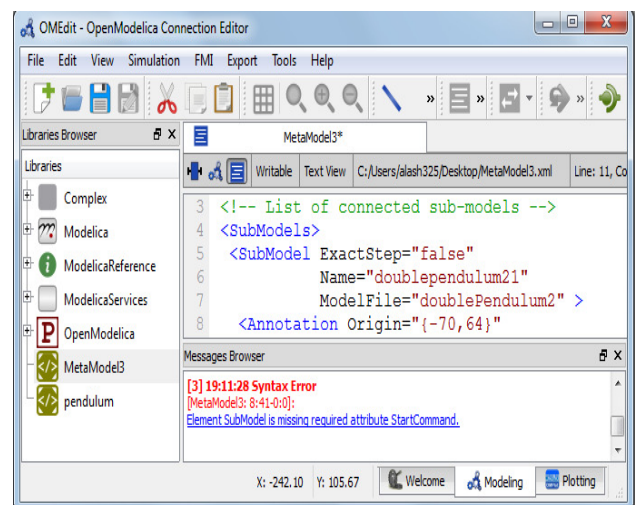


Figure 9. A screenshot of a composite modeling validation message.

5.4 OpenModelica Runtime Enhancement

To support TLM-based co-simulation the OpenModelica runtime has been enhanced. The added functionality supports single solver step simulation so that the executed simulation model can work together with the TLM manager. New flags to enable this functionality in the simulation executable are now available:

- `-noEquidistantOutputFrequency`
- `-noEquidistantOutputTime`

The new flags control the output, e.g., the frequency of steps and the time increment.

5.5 Communication with the SKF TLM Based Co-Simulation Framework

The graphic composite model editor in OpenModelica provides a graphical user interface for co-simulation of composite models. It can be launched by clicking the TLM co-simulation icon from the toolbar, see Figure 10.

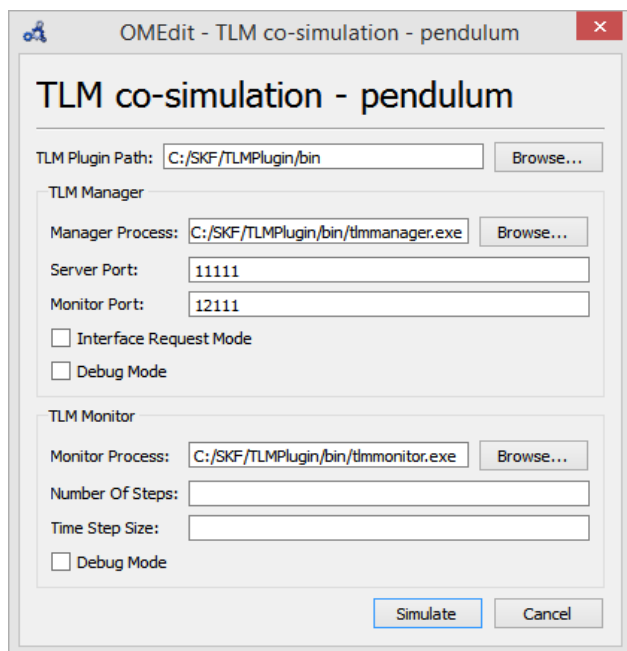


Figure 10. TLM co-simulation setup.

The editor runs the TLM simulation manager and simulation monitor. The simulation manager reads the composite model from the editor, starts the co-simulation, and provides the communication bridge between the running simulations. Figure 11 shows the running status of the TLM co-simulation.

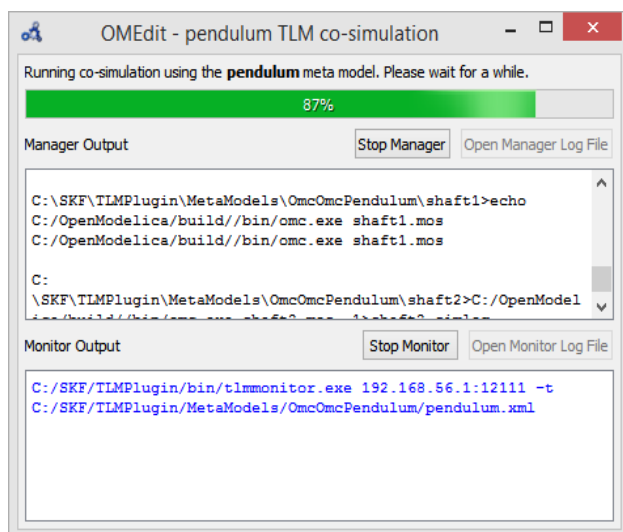


Figure 11. TLM co-simulation.

The simulation monitor communicates with the simulation manager and writes the status and progress of the co-simulation in a file. This file is read by the editor for showing the co-simulation progress bar to the user. The editor also provides the means of reading the log files generated by the simulation manager and monitor.

During the post-processing stage, simulation results are collected and visualized in the OMEdit plotting perspective as shown in Figure 12.

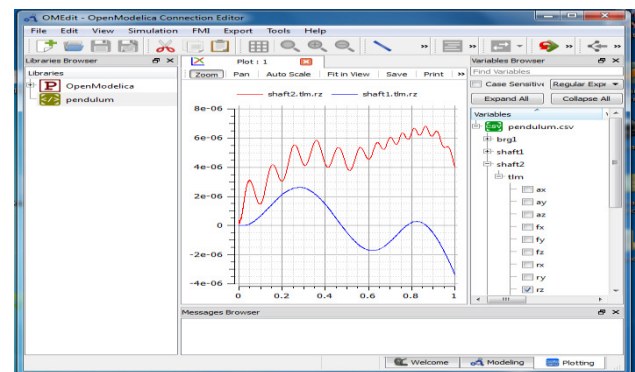


Figure 12. Results of TLM co-simulation.

6 Industrial Application of Composite Modeling with TLM Co-Simulation

SKF has successfully used the TLM co-simulation framework to simulate composite models. For example, Figure 13 shows one such application with an MSC.ADAMS (MSC-Software, 2015) car model containing an integrated SKF BEAST (Stacke, Fritzson, and Nordling, 1999) hub-unit sub-model connected via TLM-connections.

7 Conclusions and Future Work

This paper presents a general open-source graphical editor and simulation tool for composite modeling and simulation as well as its integration with SKF's TLM-based co-simulation framework for TLM based co-simulation and the OpenModelica FMI co-simulation.

The graphical editor combines a number of features to support end-users with respect to the creation of composite models and co-simulation. These include adding, removing, and connecting components (sub-models) both textually and graphically, as well as integrated co-simulation and visualization of simulation results.

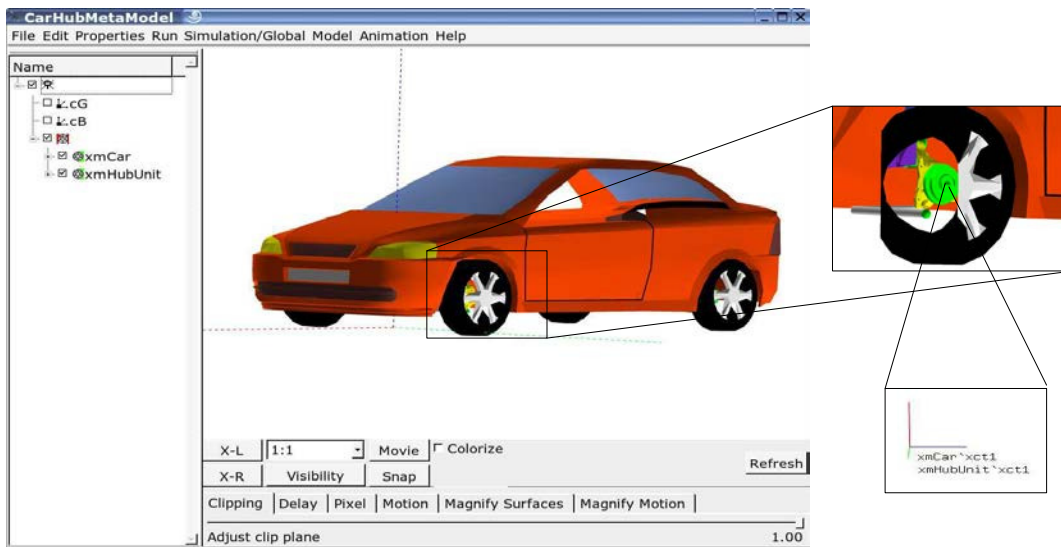


Figure 13. A composite model of an MSC.ADAMS car model with an integrated SKF BEAST hub-unit sub-model (green), connected via TLM connections for co-simulation.

The composite model editor is currently in an early stage of development but already supports external non-Modelica models represented in XML form (essentially black boxes with interfaces) inside the component tree which can be used for composite model composition.

Future development includes 3D visualization in composite modeling as well as being able to use both FMI-connections and TLM-connections in the same composite model, since they currently can only be used separately (either FMI or TLM within a specific composite model).

Future work also involves the development of a proposal for integrating TLM-based co-simulation as an option in the FMI standard, as well as participating in the standardization work in the SSP project in the Modelica Association (System Structure and Parameterization of Components for Virtual System Design abbreviated SSP), hopefully resulting in standardization of and extended version of the composite model XML schema.

A detailed comparison of TLM and FMI co-simulation is outside the scope of this paper. However, especially for mechanical applications TLM provides very simple and easy-to-use interface definitions, as well as in general numerically stable co-simulation. These are important reasons for continued use and improvement of TLM-based tools and future standardization and incorporation into the FMI standard.

Acknowledgements

The work has been supported by Vinnova in the ITEA2 MODRIO project, by EU in the INTO-CPS project, and by the Swedish Government in the Swedish Government in the ELLIIT project. The Open Source

Modelica Consortium supports the OpenModelica work. The TLM based co-simulation framework is provided by SKF.

References

- Adeel Asghar, Sonia Tariq, Mohsen Torabzadeh-Tari, Peter Fritzon, Adrian Pop, Martin Sjölund, Parham Vasaiely, and Wladimir Schamai. An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation. In *Proc. of the 8th International Modelica Conference 2011*, pp. 739–747. Modelica Association, March 2011. Linköping University, Sweden, 2010.
- David M. Auslander. Distributed System Simulation with Bilateral Delay-Line Models. *Journal of Basic Engineering, Trans. ASME*: 195–200, 1968.
- Mikael Axin, Robert Braun, Petter Krus, Alessandro dell’Amico, Björn Eriksson, Peter Nordin, Karl Pettersson, and Ingo Staack. Next Generation Simulation Software using Transmission Line Elements. In *Proceedings of the Bath/ASME Symposium on Fluid Power and Motion Control (FPMC)*, September 2010.
- Torsten Blochwitz et al. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *Proceedings of the 8th International Modelica Conference., Dresden, Mar. 2011.* doi: 10.3384/ecp11063105.
- James D. Burton, Kevin A. Edge, and Clifford R. Burrows. Partitioned Simulation of Hydraulic Systems Using Transmission-Line Modelling. In *ASME WAM*, 1993.
- FMI-Standard.org. Functional Mock-up Interface for Model Exchange and Co-Simulation Version 2.0, July 25, 2014. <https://www.fmi-standard.org/>.
- Peter Fritzon. Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach. 1250 pages. ISBN 9781-118-859124, Wiley IEEE Press, 2014.

- HOPSAN. The HOPSAN Simulation Program, User's Manual. *Linköping University, 1985*. LiTH-IKP-R-387.
- Peter B. Johns and Mark O'Brien. Use of the transmission line modelling (TLM) method to solve nonlinear lumped networks. *The Radio and Electronic Engineer*, 50(1/2):59–70, 1980.
- Petter Krus, Arne Jansson, Jan-Ove Palmberg, and Kenneth Weddfelt. Distributed Simulation of Hydromechanical Systems. In *Proc. of the Third Bath International Fluid Power Workshop*, 1990.
- MSC-Software, MSC.ADAMS – interactive motion simulation software, <http://www.mscsoftware.com> (accessed: 22th of May 2015).
- Iakov Nakhimovski. Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, *Dissertation No. 1009*, Linköpings universitet, Sweden, 2006.
- Alexander Siemers, Iakov Nakhimovski, and Dag Fritzson. Meta-modelling of Mechanical Systems with Transmission Line Joints in Modelica. In *Proceedings of the 4th International Modelica Conference*, Hamburg, Germany, 2005.
- Alexander Siemers, Peter Fritzson, and Dag Fritzson, Meta-Modeling for Multi-physics Co-simulations applied for OpenModelica. In: *Proc. of ANIPLA 2006 International Congress on 'Methodologies for Emerging Technologies in Automation'*, University of Rome La Sapienza, November 13–14–15, 2006.
- Alexander Siemers and Dag Fritzson. A meta-modeling environment for mechanical system co-simulations. In *Proc. of the 48th Scandinavian Conference on Simulation and Modeling (SIMS 2007)*, Gothenburg (Särö), Sweden, October 2007.
- Alexander Siemers, Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis, *Dissertation No. 1337*, Linköpings universitet, Sweden, 2010
- Lars-Erik Stacke, Dag Fritzson, and Patrik Nordling, BEAST—A Rolling Bearing Simulation Tool, Proc. Instn Mech. Engrs, part K, *Journal of Multi-body Dynamics*, 1999.

The Modelica language and the FMI standard for modeling and simulation of Smart Grids

Olivier Chilard¹ Jérémy Boes² Alexandre Perles² Guy Camilleri² Marie-Pierre Gleizes²

Jean-Philippe Tavella¹ Dominique Croteau¹

¹EDF Research and development, 1 avenue du général de Gaulle, 92140 Clamart France
{olivier.chilard, jean-philippe.tavella, dominique.croteau}@edf.fr

²Institut de Recherche en Informatique de Toulouse (IRIT), SMAC, Toulouse University, Université Paul Sabatier, France,
{Jeremy.Boes, Alexandre.Perles, Guy.Camilleri, Marie-Pierre.Gleizes}@irit.fr

Abstract

The smart power grids will extensively rely on network control to increase efficiency, reliability, and safety. In this context, the simulation of such complex systems is becoming an essential tool to support the development of Smart Grids.

This paper presents an overview of the EDF R&D Modelica library **GridSysPro** (GSP), which provides electrical components adapted to Smart Grid simulation; and a multi-agent approach for supporting the co-initialization process of complex network of FMUs.

Keywords: Smart Grid, Co-Simulation, Modelica.

Introduction

The smart power grids will extensively rely on network control to increase efficiency, reliability, and safety; to enable plug-and-play asset integration, such as in the case of distributed generation and alternative energy sources; to support market dynamics as well as reduce peak prices and stabilize costs when supply is limited. In turn, network control requires an advanced communication infrastructure with support for safety and real-time communication (Figure 1).

Simulating such complex systems is required for the development of Smart Grids. Several simulation tools are available on the market but these tools have two major drawbacks:

- They are generally not designed to import models developed for other tools.
- They are not adapted to large scale complex system of systems or cyber-physical systems as smart grids which require time-consuming calculation.

One solution to bypass these drawbacks is to use a co-simulation platform which can connect together several simulators and FMUs (Functional Mock-up unit).

EDF R&D is funding the development of its own co-simulation platform dedicated to the Smart Grids in partnership with LORIA-INRIA. A first release of this tool named MECSYCO is available under the Affero

GPL license v3 (<http://mecsyclo.loria.fr/>). The next published version (at the end of 2015) will upgrade MECSYCO with the coupling of different types of discrete-time or continuous-time simulators (including the FMUs) divided in three domains:

- The physics domain (process) : FMUs exported according to the FMI 2.0 standard from Dymola with models built from the EDF Modelica library **GridSysPro** or historical tools widely used at EDF (e.g. EMTP-RV) now compatible with the FMI standard;
- The telecommunication domain: NS-3, OMNeT++ or OPNeT ;
- The Information System domain with models designed with UML/SysML oriented tools.

MECSYCO is based on the Multi-Agent concept (one agent per simulator to describe a heterogeneous multi-model) and on the DEVS formalism (to conceive a decentralized execution algorithm respecting the causality constraints).

This paper provides first an overview of the EDF R&D Modelica library **GridSysPro** (GSP) composed of electrical components mapped on the zone related to the process of a Smart Grid (Figure 1). Besides that, to comply with the modeling of large scale electrical networks, a solution to co-initialize several interconnected FMUs exported from Dymola is described.

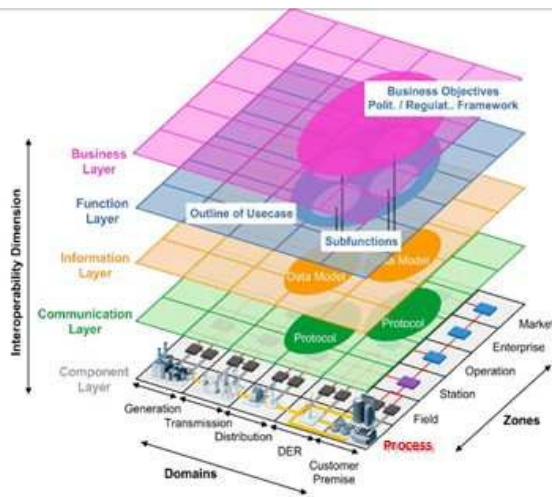


Figure 1 : The Smart Grid Architecture Model (SGAM)

1 GridSysPro Library

Modeling of electrical networks has always been a major scientific challenge for analysis and design.

Models are often used for studies of stability and control, for the analysis and optimization of power flow or for harmonic analysis and their distortion.

The common approach in electrical network simulation is based on classification of the phenomena according to their time scales (Figure 2). For each class of phenomena, particular mathematical models are developed (Figure 3).

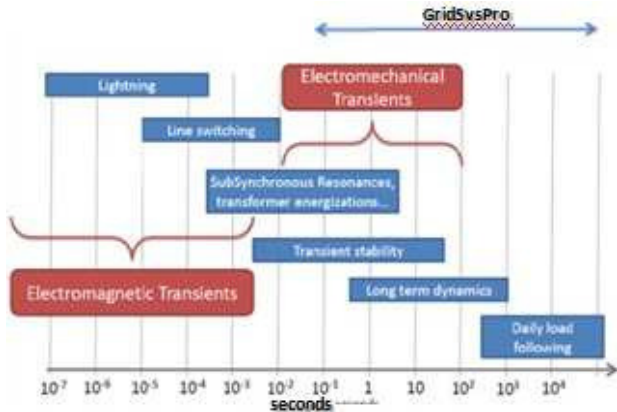


Figure 2 : Power system dynamics

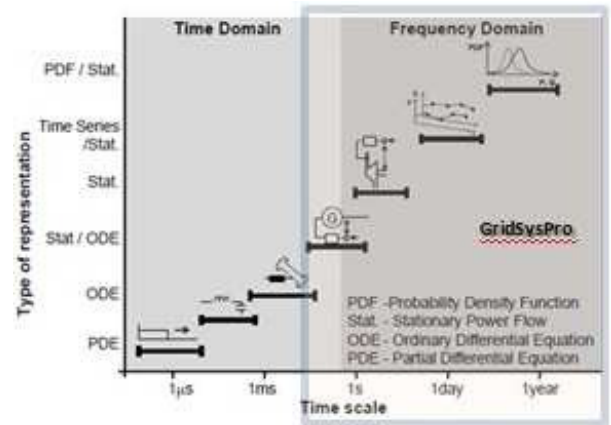


Figure 3 : Model representations for different time scales

1.1 Objectives of GSP

GridSysPro (GSP) is a Modelica library which allows stationary power load flow calculation, short circuit analyses and transient stability simulations.

The goal of stationary power load flow analysis is to find all branch currents and all nodal voltages amplitude and their angles according to electrical constraints applied at each injection node. It can help to calculate the use of power system resources and the power quality with respect to the voltage bandwidth constraints. In the real world, such analysis may be done for anticipating the effects of future operation decisions. In the simulated Smart Grid, the power flow analysis is a vital function to get the line currents and node voltages in the real power system. With this information, compliance to operating limitations such as those stipulated by voltage ranges and maximum loads, can be examined. In this way, the location of congestions and power outage situations can be identified. Moreover, the stationary power flow analysis is required to help the self-healing function, after the isolation step of the faulted section, to re-establish service to as many customers as possible from alternative sources/feeders in accordance with the operating limitations. Due to the ability to determine losses and reactive-power allocation, load-flow calculation also supports the planning engineer in the investigation of the most economical operation mode of the network.

Short circuit analysis recalculates the power flow after the occurrence of a fault in a power network. The faults may be a three-phase short circuit, a one-phase grounded, a two-phase short circuit, a two-phase grounded, a one-phase break, a two-phase break or a more complex fault.

The goal of transient stability simulation of power systems is to analyze the stability of a power system in a time window of a few seconds to several tens of seconds. Stability in this aspect is the ability of the system to quickly return to a stable operating condition after being exposed to a disturbance such as for example a tree falling over an overhead line resulting in the automatic disconnection of that line by its protection

systems. In engineering terms, a power system is deemed stable if the rotational speeds of motors and generators, and substation voltage levels can return to their normal values in a quick and stable manner.

1.2 Overview of the GridSysPro library

According to the objective retained, GSP allows the modeling of both transmission (HV) and distribution (MV/LV) electrical networks. The first version of GSP provides the following components:

- lines,
- transformer with or without load tap changer and different winding coupling,
- generators,
- adapted blocks in order to build different types of controllers like voltage and speed regulators,
- generic load which can represent different types of consumption according to sensitive factor as parameters related to voltage and frequency,
- electrical faults, analysers and breakers.

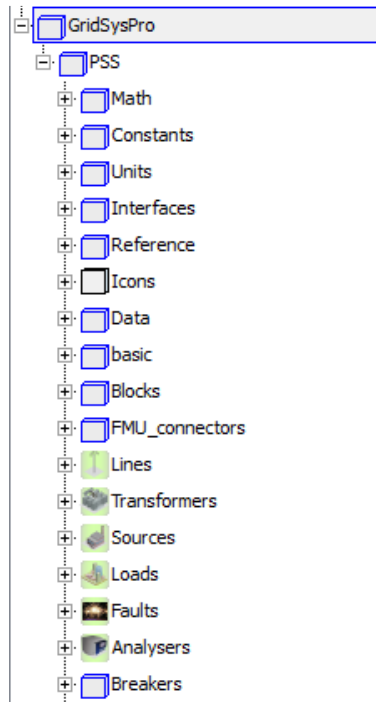


Figure 4 : Packages of GSP

Data and an Icon, which correspond respectively to external parameters and a graphical representation, are inherited by each main electrical component.

Component models are stored in hierarchically structured packages. The blue ones provide all elementary functions and models required to describe the main components (green) needed for network modeling.

1.3 Principles retained for the development of GSP

Because of electromagnetic transients are not considered in the GSP development (Figure 2 and Figure 3), power systems are described in a form using system of algebraic-differential equations. Thus the behavior of each passive component of the grid is defined by algebraic equations (complex number formulations) while the one related to electrical and mechanical parts of machines are determined by a system of differential equations.

In order to simulate large-scale three phases balanced and unbalanced networks, passive components have been defined by three single phase Quadruples $Y(QY)$.

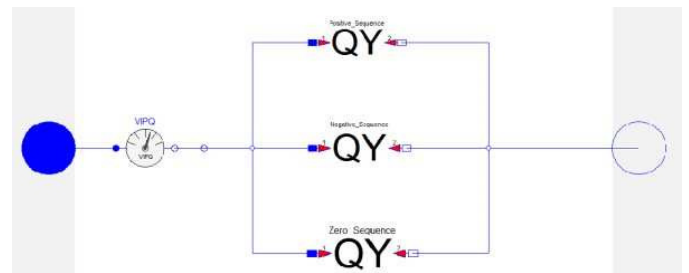


Figure 5 : model of a passive component

The algebraic equation of a QY is defined according to the equation (1) where only the variables of the positive and negative pins (p, n) are considered.

$$\begin{bmatrix} p.i \\ n.i \end{bmatrix} = [y] \cdot \begin{bmatrix} p.v \\ n.v \end{bmatrix} \quad (1)$$

Thus GSP passive components are described by three generic QY objects connected to two composite connectors containing three pins. The latter are related respectively to the positive, negative and zero sequence circuits. More precisely, the three phases a, b and c of each passive component are broken down into three sets of balanced single-phase phasors 1, 2 and 0 according to the transformation of Fortescue (2).

$$\begin{bmatrix} I_{abc}^p \\ I_{abc}^n \end{bmatrix} = \begin{bmatrix} F & 0_{3 \times 3} \\ 0_{3 \times 3} & F \end{bmatrix} \cdot P^{-1} \cdot \begin{bmatrix} Y_1 & 0_{2 \times 2} & 0_{2 \times 2} \\ 0_{2 \times 2} & Y_2 & 0_{2 \times 2} \\ 0_{2 \times 2} & 0_{2 \times 2} & Y_0 \end{bmatrix} \cdot P \cdot \begin{bmatrix} F & 0_{3 \times 3} \\ 0_{3 \times 3} & F \end{bmatrix}^{-1} \begin{bmatrix} V_{abc}^p \\ V_{abc}^n \end{bmatrix} \quad (2)$$

with:

$$\begin{bmatrix} I_a \\ I_b \\ I_c \end{bmatrix} = [F] \cdot \begin{bmatrix} I_1 \\ I_2 \\ I_0 \end{bmatrix} \quad \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = [F] \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_0 \end{bmatrix}$$

$$\begin{bmatrix} I_1^{pn} \\ I_2^{pn} \\ I_0^{pn} \end{bmatrix} = P \cdot \begin{bmatrix} I_{120}^p \\ I_{120}^n \end{bmatrix}$$

Therefore, the components of a grid, as lines, cables and transformers are represented by three decoupling circuits (Figure 5). Thus, the behavior of each object differs only by the definition of the y-parameter matrix $(\underline{Y}_1, \underline{Y}_2, \underline{Y}_0)$ of each QY model. Besides that an analyzer VIPQ can be used in order to provide voltage, current and power flow per phase (a, b, c).

During the initialization which is equivalent to a load flow calculation:

- generators are represented as either PQ or PV node as slack node depending on the attribute: LoadFlow_type. For each dynamic state variable on which derivative is applied one equation is given in the initial equation section,
- loads are defined as constraint of consumption.

1.4 Illustrations and validations

Some validations of GSP have been done where load flow and dynamic behaviors have been tested and compared respectively with OpenDSS and ObjectStab.

The OpenDSS is an electrical power system simulation tool developed by EPRI (USA Electric Power Research Institute) primarily for electric utility power distribution systems. It supports nearly all frequency domain (sinusoidal steady-state) analyses commonly performed on electric utility power distribution systems.

The ObjectStab package is a free Modelica Library for power systems voltage and transient stability simulations limited to single phase description of Network and dedicated to students. For GSP validation the use case of ObjectStab validated with EUROSTAG (common tool used by utilities for transient stability simulations) has been retained.

All these tests have been successful and the one related to the load flow simulation is presented here after.

The considered MV Network is a typical outgoing feeder of EDF energized by its MV primary substation (Figure 6). In order to simplify the description of networks a **Network Management Tool** developed by EDF R&D under MATLAB has been used. This NMT allows an automatic generation of the Modelica model of a network from the CIM XML file. More precisely the IEC 61970/61968 (CIM) provides a Common Information Model to support the information exchange between different EMS (Energy Management System). Its large data model provides the possibility to model physical (like cables, switches) and abstract objects (like documents, schedules, and consumer data) in the energy domain. The databases of EDF’s electrical networks have been built according to the CIM standard. Therefore NMT and Dymola/GridSysPro allow an automatic Modelica implementation of EDF grids.

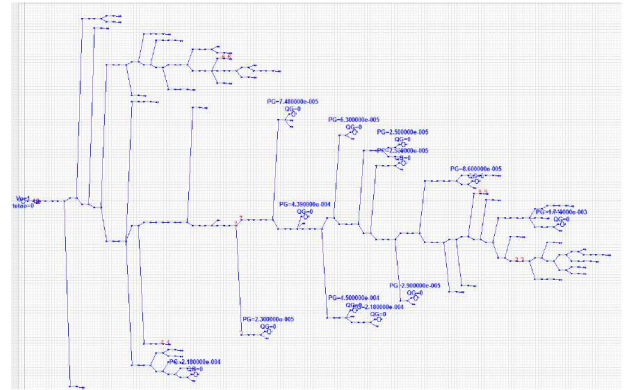


Figure 6 : the outgoing MV feeder retained for the GSP test

The Load flow simulations results are provided in Figure 7. These latter correspond to the voltage amplitude profile along the considered MV network from the primary substation to the end points of the grid.

The results obtained respectively by OpenDSS and GridSysPro are identical.

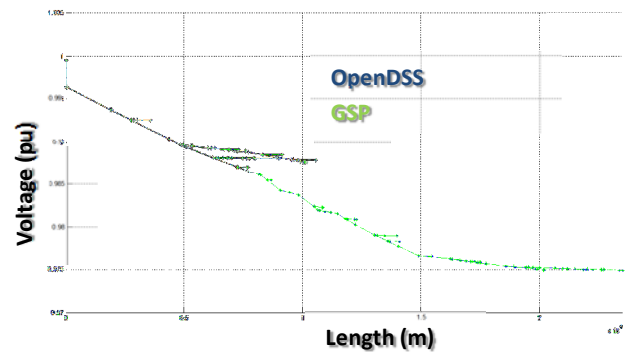


Figure 7 : The voltage amplitude profile obtained with OpenDSS and GSP

2 Co-initialization with FMUs exported from GSP

One solution to comply with the simulation of very large scale electrical networks described from GSP is to export the latter as several FMUs to be interconnected and simulated inside a co-simulation platform like MECASYCO. The segmentation of electrical networks into a set of FMUs is a design choice that depends on models and solvers properties (their execution cost...).

However, in this case the load flow calculation is distributed into each FMU and a master has to be developed in order to coordinate the calculation of each boundary variables related to input and output of each FMU.

2.1 Co-initialization with an Adaptive MultiAgent System (AMAS)

A graph of connected FMUs can be expressed in general way as following $(out) = \langle FMU \rangle (in)$. Each

FMU can be viewed as a function owning inputs and providing outputs. FMUs are connected through some of their inputs assigned to outputs of other FMUs, defining the calculation graph of the global problem. In this view, the vectors \overrightarrow{in} and \overrightarrow{out} represent all inputs and outputs of all FMU, while $\langle FMU \rangle$ is the aggregation of all FMU functions on which the input vector \overrightarrow{in} is applied. Trying to co-initialize multiple FMU is equivalent to verifying the following property: $\overrightarrow{out} = \langle FMU \rangle(\overrightarrow{in}) = \overrightarrow{in}$.

Thus, the co-initialization problem is in its general formulation equivalent to the search of fixed points in mathematics. Indeed, given a set E and an application $f: E \rightarrow E$, a point x is a fixed point if $f(x) = x$. E can be a metric space in n dimensions. When the FMUs graph owns many cycles, the problem of co-initialization corresponds to a complex fixed point search problem.

In this paper, we propose to explore the potentialities of a multi-agent approach for solving this type of fixed point search problem. For this, we choose to apply the AMAS (Adaptive Multi-Agent Systems) theory developed by SMAC team in (Georgé, Gleizes, & Camps, 2011). This theory has shown its suitability for solving complex and dynamic problems in many applications (Jorquera, Georgé, Gleizes, & Régis, 2013), (Brax, Andonoff, Gleizes, & Glize, 2013), (Capera, Gleizes, & Glize, Mechanism Type Synthesis based on Self-Assembling Agents, 2004).

In this section, we will very briefly present some important concepts of Multi-Agent Systems and the AMAS theory. The AMAS theory will then be used as a guide in the design of a multi-agent system able to solve the fixed point search problem presented above. Finally, we will present some results of the application of this multi-agent system on a GSP generated case study.

2.2 Multi-Agent Systems

A multi-agent system is a set of autonomous entities called agents, interacting in a common environment, acting to solve in coherent way a common task. This last point is important because it implies the unity of the MAS. Even if each agent has its own individual goal, in some situations their goal can possibly be in conflict with the others.

According to (Wooldridge & Jennings, 1995) and (Ferber, 1999), an agent is a physical or a software entity which:

- is autonomous,
- exists in an environment that it can perceive and on which it can act,
- has a partial representation of this environment,
- is able to communicate with other agents,
- has resources,

- has skills and can offer services.

The behavior of an agent results from its perceptions, its knowledge, its skills, and naturally its goals. It follows a life cycle in three stages repeated infinitely throughout its execution:

- the stage of *perception* during which the agent acquires new information on the environment,
- the stage of *decision* in the course of which the agent chooses the next actions to be made,
- the stage of *action* during which the agent performs the actions chosen in the previous stage.

An essential characteristic of agents is their autonomy: they decide themselves to act or not and the nature of their actions.

2.3 Adaptive Multi-Agent System Theory

The Adaptive Multi-Agent System (AMAS) theory appears suitable for the fixed point search problem (see (Capera, Georgé, Gleizes, & Glize, 2003) (Whitehead, 2008)).

Due to their distributed structure, AMAS are flexible and self-adaptable to several strategies of simulators control. The first aim of the AMAS theory is to design Multi-agent System having a coherent collective activity that achieves the right task. This property is named "functional adequacy" and the following theorem is proved: "For any functionally adequate system, there is at least a cooperative interior medium system which fulfills an equivalent function in the same environment". Therefore, it focuses on the design of cooperative interior medium systems in which agents are in cooperative interactions. The specificity of the theory: "the emergence" resides in the fact that the global function of the system is not coded within the agents. Agents have only a partial knowledge. The global function of this system emerges from the collective behavior of the different agents composing it. Each agent possesses the ability of self-organization i.e. the capacity to locally rearrange its interactions with others depending on the individual task it has to solve. Changing the interactions between agents can indeed lead to a change at the global level. This induces the modification of the global function. This capacity of self-organization enables to change the global function without coding this modification at the upper level of the system. Self-organization in AMAS is based on the capacity an agent possesses to be locally "cooperative".

Therefore AMAS agents locally cooperate in order to satisfy their own goals as well as they try to help other agents to achieve their goals. This notion of local goals is crucial for reaching a global solution, and is represented by a measure of criticality. This measure denotes the agent difficulty to reach its goals. It is used in a local way by agents in order to result in a system where the satisfaction of all agents is balanced.

Moreover, an agent will modify its behavior if it thinks that its actions are useless or detrimental to its environment. Such situations are called Non-Cooperative Situations (NCS). Some behavioral rules, specific to NCS's, help agents to solve or avoid these situations. By solving NCS's, in regard to their own local goals, cooperative agents collectively find a solution to the global problem. Therefore one can consider the behavior of an AMAS as emergent.

2.4 AMAS for co-initialization (Fixed Point Search Problem)

In this part, we present the use of the AMAS theory as a guide for the design of a multi-agent system dedicated to the co-initialization (solving the fixed point search problem). Following the AMAS theory, we start by identifying agents, their neighborhood, their criticality and their perceptions and actions. After, we will very roughly describe the behavior of agents.

2.4.1 Agents

The objective of the co-initialization is to reach a state of the FMUs graph in which every input of every FMU is equal to the output of other FMU connected in input. We chose to represent in the form of agent the connections between FMUs. In other words, if an agent represents one link between only two FMUs then the extremities of the link must be equal. More exactly, we place an agent at every output of every FMU, as in Figure 8

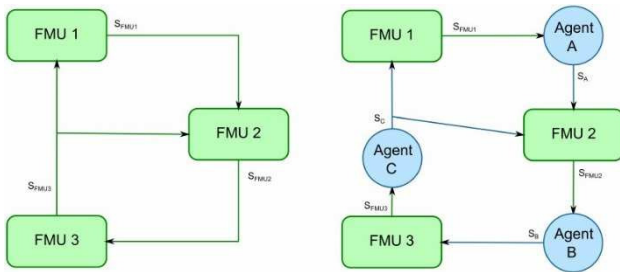


Figure 8 To the left, an example of FMU graph. To the right, the same graph with cooperative agents.

In this figure, the following equalities have to be satisfied: $S_{FMU1} = S_A$, $S_{FMU2} = S_B$ and $S_{FMU3} = S_C$. Therefore, the proposed system is only composed of one agent type corresponding to links between FMUs.

2.4.2 Neighborhood

The neighborhood of an agent is defined as the set of all agents being directly influenced by it. Therefore, the neighborhood of an agent α corresponds to all agents in output of the FMU to which α is connected in input, as well as the agent α itself. For example (Figure 6), the neighborhood of the agent A is composed of agent A and B whereas the neighborhood of the agent C consists of three agents A, B and C.

2.4.3 Criticality Measure

The fixed point search problem is solved if, after having acted, every agent observes on its inputs a value equal to the one that it had assigned on its outputs in the previous step. In other words, the problem is solved if, for every agent: $|input_t - output_{t-1}| = 0$ which constitutes the own objective of all agents of this system.

The criticality represents the difficulty that an agent has to satisfy its own objective. In this case, the criticality measure is obvious: $criticality = |input_t - output_{t-1}|$. Following AMAS theory, all agents will try to decrease their criticality to 0 what will solve the fixed point search problem.

2.4.4 Perceptions and Actions

An agent A corresponding to the output S of a FMU, perceives on its input the value of S. A also perceives the values of criticality of all agents of its neighborhood. Finally, A perceives the value of partial derivatives (Jacobian matrix) of all FMU functions to which it is connected.

The agent A can modify its own output, which is assigned a value to all FMU inputs to which S is connected. The agent action can thus be of three types: increase, decrease, or not change its value of output.

2.4.5 Agent Behavior

Our system is homogeneous, meaning that all agents possess the same behavior algorithm. The objective of each agent is to decrease the level of criticality of its neighborhood (including itself). The action of an agent can have a beneficial effect (which imply a decrease of the criticality), harmful (which causes an increase of the criticality), or indifferent (which does not provoke a variation of criticality) on each agent of its neighborhood.

Thanks to the observation of the sign of Jacobian matrix of all FMU connected to its output, each agent can, to a certain extent, know the effect of its action.

An agent has to form an idea of the amplitude and the direction in which it will vary the value of its output in order to decrease the criticality of its neighborhood as fast as possible. This information is represented by two internal variables, managed dynamically:

- δ is a positive real value corresponding to the amplitude of the variation,
- σ is an integer in $\{-1;0;1\}$, it indicates the direction of the variation.

At each life cycle, an agent modifies its output value in the following way:

$$o_t = o_{t-1} + a_t$$

Where a_t is calculated from the amplitude σ and the direction δ . Indeed, we apply a variation in the direction σ with amplitude equal to δ . Thus :

$$a_t = \begin{cases} \sigma \cdot \delta & \text{if } \sigma \neq 0 \\ \text{random}(-\delta, \delta) & \text{else} \end{cases}$$

If $\delta=0$, the variation is randomly decided between $-\delta$ and δ , both values being equiprobable. The presence of random can be justified by the fact that all agents perceive, decide and act simultaneously by following the same behavior. If the latter was purely determinist, the system would have to cope with the problem of the bar of El Farol (Whitehead, 2008), resulting in a non-desirable synchronization of actions, and thus an ineffective exploration of the search space and a convergence outside the solution.

According to their perceptions and following cooperative rules, agents adjust their δ and σ variables in order to decrease the criticality of themselves and their neighbors. Thus, the overall criticality tends to decrease over time, while the AMAS converges toward a solution. Due to the size of this paper, we will not describe the adjustment mechanism of agents.

2.4.6 A case study: FMU graph generated with GSP

In this case study, we consider three FMU: A, B and C having respectively two, four, and two outputs. Thus there are 8 agents in the AMAS system which will initialize the network. In the Figure 9, the neighborhood graph of agents is presented. Agents are assigned to outputs of the considered FMU network. Each FMU was exported according to the FMI 2.0 standard from Dymola with EDF Modelica library GSP.

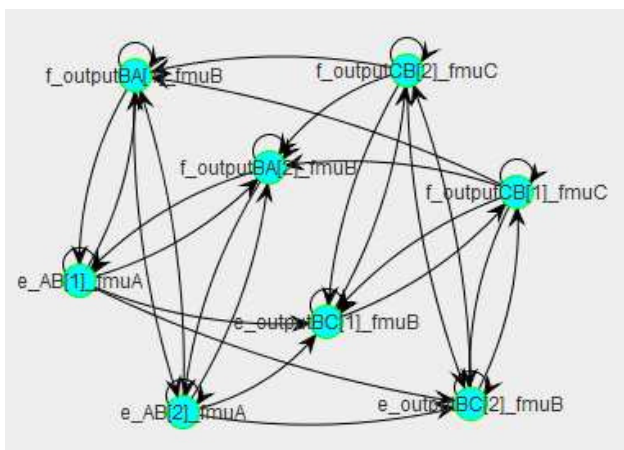


Figure 9: Neighborhood Graph

From a theoretical point of view, for solving the fixed point search problem, the criticality of all agents should reach the value 0. However, in practice, due to numerical aspects, it may be reasonable to reach a value close to 0. In this case study, the fixed point search was stopped if agent criticalities reach a value lower than 10^{-4} .

The Figure 10, shows the criticality curves of the eight agents. The total number of system cycles is indicated in abscissa and criticality values in ordinate. The best

solution is reached after around 49000 cycles, with a residue (error) of 3.74×10^{-5} .

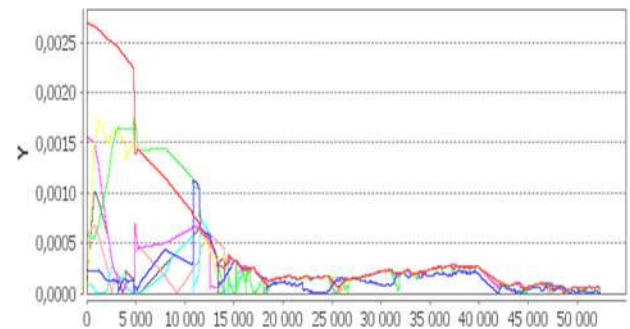


Figure 10: AMAS System Convergence

In this case study, the AMAS system converges with low criticality values (lower than 10^{-4}). Therefore, the presented system is able to co-initialize the FMU network with a reasonable precision.

Moreover, the AMAS algorithm was previously used in quite different domains with several thousands of agents (corresponding here to the number of parameters of the FMUs). The resolution principle is totally local and depends mainly on two characteristics of the application:

1. The number of agents influencing quasi-linearly the solving time,
2. the branching factor the agents (equivalent to the mean number of its neighbors). For a given class of problem (here co-initialization of FMUs), the number of cycles is stable.

Consequently the duration of the co-initialisation problem depends roughly linearly of the number of FMU.

3 Conclusions and perspectives

This paper presents an overview of the Modelica library **GridSysPro** (GSP) composed of electrical components mapped on the zone related to the process of a Smart Grid. Beside that to comply with the modeling of large scale electrical networks a solution to co-initialize several interconnected FMUs exported from GSP/ Dymola, is described. More precisely the interconnection of several FMUs requires the determination of initial values of all FMU inputs (co-initialization). This problem is complex and can be formulated as a fixed point search problem. We proposed the use of the AMAS (Adaptive Multi-Agent System) theory for designing a system able to solve this problem. We illustrate the suitability of the proposed system in a case study generated from GSP.

The previously presented version of **GridSysPro** includes several components allowing it to represent and simulate an electrical network. Nowadays, we are moving toward the concept of Smart Grid which is an evolution of the electrical network allowing notably bi-directional exchanges of energy and information through lines and an intelligent and autonomous control.

That is why an interesting perspective could be to integrate a set of advanced features to this library as part of the initiative on FMU, co-initialization and Modelica.

As an example, the first advanced feature that will be integrated is an autonomous voltage regulation system. This feature is expected, on the one hand, to be able to build a coordinated regulation between medium voltage and low voltage networks and, on the other hand, to deal with the massive integration of decentralized generators.

As previously seen, the Adaptive Multi-Agent System theory seems adapted to solve this kind of problem, notably by the amount of elements interacting in the system and by the need to support the topology changes.

The proposed approach splits the problem into two steps. Firstly, informed agents get the voltage and power values from sensors they are linked with and cooperate with others in order to help them find the missing values. And secondly, agents communicate in order to find the set of voltage set-points to guarantee the compliance with the contractual voltage range at consumption points. Such an approach of this problem allows building a voltage regulation regardless of the size of the network.

4 References

- Julien Vaubourg, Yannick Presse, Benjamin Camus, Christine Bourjot, Laurent Ciarletta, Vincent Chevrier, Jean-Philippe Tavella, Hugo Morais, Boris Deneuve, & Olivier Chillard (PAAMS 2015). SmartGrid Simulation with MECSYCO.
- Brax, N., Andonoff, E., Gleizes, M.-P., & Glize, P. (2013). Self-adaptive Aided Decision-making - Application to Maritime Surveillance. *International Conference on Agents and Artificial Intelligence (ICAART)*, (pp. 419-422). Barcelona: INSTICC - Institute for Systems and Technologies of Information, Control and Communication.
- Capera, D., Georgé, J.-P., Gleizes, M.-P., & Glize, P. (2003). The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. *International Workshop on Theory And Practice of Open Computational Systems*, pp. 389-394.
- Capera, D., Gleizes, M.-P., & Glize, P. (2004). Mechanism Type Synthesis based on Self-Assembling Agents. *Journal of Applied Artificial Intelligence*, 921-936.
- Ferber, J. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley Reading.
- Georgé, J.-P., Gleizes, M.-P., & Camps, V. (2011). Cooperation. (G. Di Marzo Serugendo, M.-P. Gleizes, & A. Karageorgos, Eds.) *Self-organising Software*, 193--226.
- Jorquera, T., Georgé, J.-P., Gleizes, M.-P., & Régis, C. (2013). A Natural Formalism and a MultiAgent Algorithm for Integrative Multidisciplinary Design Optimization. *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), Atlanta, USA, 17/11/2013-20/11/2013* (pp. 146 - 154). IEEE Computer Society.
- Whitehead, D. (2008). The el farol bar problem revisited : Reinforcement learning in a potential game. *ESE Discussion Papers*, 186.
- Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2), 115-152.

Coupled modeling of a District Heating System with Aquifer Thermal Energy Storage and Absorption Heat Transformer

Carles Ribas Tugores¹ Henning Francke² Falk Cudok³

Alexander Inderfurth¹ Stefan Kranz² Christoph Nytsch-Geusen¹

¹Fachgebiet für Versorgungsplanung und Versorgungstechnik, Berlin University of the Arts, Germany, {c.ribastugores, a.inderfurth}@udk-berlin.de

²Helmholtz Centre Potsdam - GFZ German Research Centre for Geosciences, {kranz, francke}@gfz-potsdam.de

³Institute of Energy Engineering, Technische Universität Berlin, Germany, falk.cudok@tu-berlin.de

Abstract

Aquifer thermal energy storages (ATES) are a promising technology for seasonal thermal energy storage which can bridge the gap between constant production and seasonally varying demand. This paper presents first simulation results of an energy concept proposed for the university campus Berlin-Charlottenburg, which is characterized by the combination of an ATES system as a seasonal thermal energy storage and an absorption heat transformer (AHT), which supplies 50 buildings of the campus with heating energy. Furthermore, the paper deals with the modeling of the different subsystems, described in Modelica; energy production, storage, consumption and distribution and their integration in a coupled Modelica system model.

Keywords: Modelica, ATES, geothermal, absorption heat transformer, district heating, FVM

1 Introduction

Some thermal energy systems such as the ones including combined heat and power plants (CHP), which provide electrical base loads or solar thermal energy show a strong time mismatch between consumption and production. This mismatch can be compensated by the addition of a thermal energy storage (TES) to the system (see Figure 1).

The selection of a TES mainly depends on the required storage period, the economic viability and the operating condition (Dinçer & Rosen 2002). Among the possibilities for low-medium temperatures (10...90 °C), the sensible heat storage tank with water is the most common choice (Dinçer & Rosen 2002). This is mainly due to their simplicity, low cost, good performance and the favorable thermal properties of water, namely high specific heat capacity and relative high density. However, this solution is designed to compensate the daily variation. If seasonal variations are to be compensated, a larger storage is needed. In this scenario the utilization of natural aquifers as

seasonal thermal energy storage (ATES) is an interesting possibility with low cost in relation to its high storage capacity.

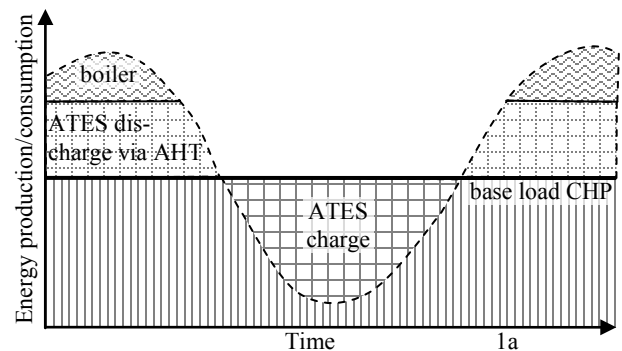


Figure 1. Seasonally fluctuating energy demand (dashed) covered by combined heat and power plant (CHP, solid) and previously stored CHP surplus recovered from aquifer thermal energy storage (ATES), topped up by a boiler.

An ATES uses groundwater in an aquifer to store thermal energy for several months, mainly for heating and cooling of buildings. The aquifer is an underground layer where water can flow through the permeable material. Common ATES systems consist of two wells (or well groups), a cold well and a warm well. When charging the ATES system with heat, groundwater is produced from the cold well, heated up and re-injected into the aquifer through the warm well. To discharge the ATES, the flow direction of water is reversed and heat is extracted (see Figure 2). The heat is typically recovered at a temperature lower than the original injection temperature due to heat exchange with the porous matrix and adjacent layers.

In order to avoid triggering the precipitation of dissolved minerals and potentially reducing aquifer permeability, the injection temperature should be kept below a certain limit T_{inj}^{max} , which depends on the fluid chemistry. This upper temperature limit does not always meet the requirements of the consumers,

making it necessary to raise the temperature to suitable temperatures for the existing heating systems. In this regard the absorption heat transformer (AHT) is an appropriate technological solution that is able to use thermal energy at an intermediate temperature level to heat a fluid at a high temperature level.

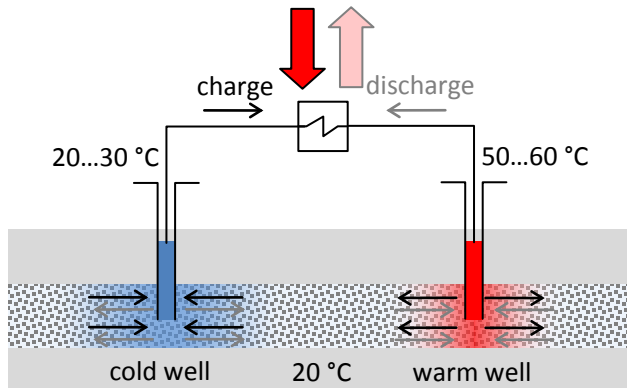


Figure 2. ATES principle with two wells.

The ATES technology has already been implemented in several projects such as in Rostock (Schmidt et al. 2000) or the German parliament buildings in Berlin (Sanner et al. 2005). There is, however, still potential to improve this technology itself as well as a need to investigate its match with other thermal technologies. Hence, the main goal of the joint project "Efficiency and reliability of energy systems in urban districts with seasonal energy storage in aquifers", is the development of a design concept, regardless of location, that assists the designer in the implementation of feasible and efficient ATES systems.

This paper presents the first results of the simulations of an energy concept proposed for the university campus Berlin-Charlottenburg and a brief description of the implemented models in Modelica. The main feature of the system is the combination of an ATES system as a seasonal thermal energy storage with an AHT.

1.1 Joint Project

The interdisciplinarity of the topic requires the participation of different research groups. Research groups from Helmholtz Centre Potsdam - GFZ German Research Centre for Geosciences (GFZ), Technische Universität Berlin (TUB) and Berlin University of the Arts (UdKB) are involved in the project in order to properly cover its different aspects.

The International Centre for Geothermal Research at GFZ covers research in a holistic approach along the whole chain of geothermal technologies from the geothermal reservoir to the provision of power, heat, and chill. One part of this work deals with simulation, evaluation and design of ATES systems.

At the department of Energy Engineering of TUB the work focuses on the development of an absorption device, which features a great flexibility in terms of

operating modes. It can work as absorption heat pump, absorption chiller (heat pump type I) or absorption heat transformer (heat pump type II). For that purpose, an experimental set-up is currently being installed that allows a better understanding of this technology. Furthermore, theoretical models of this absorption device are implemented in Modelica and presented here.

At the research group of UdKB the focus is on the energy consumption of urban districts, the energy supply systems and the provision with renewable energy. One main task is the development of a method to model a whole district, including buildings and installations, renewable energy production, as well as the distribution networks for cooling and heating, with strongly simplified low-order building models that allow to study such complex systems with low computational effort.

Besides the tasks described above, all project partners collaborate on the presented case study, the university campus Berlin-Charlottenburg, which on one hand serves to face the challenges of a real-world project, such as the data collection related to the buildings and the obtainment of a drilling license. On the other hand, different energy concepts can be studied based on a realistic scenario.

1.2 Energy Concept

From a thermal point of view, the presented system can be divided into three temperature ranges, at which the different subsystems either extract or inject heat: high temperature (above 70 °C), intermediate temperature (between 40 and 70 °C) and ambient temperature (see Figure 3). Thermal energy at the high temperature range is produced by combined heat and power plants (CHP), boilers and the AHT. This thermal energy is used to supply the buildings via a district heating network (DHN). The intermediate temperature range includes the hot well of the aquifer and the return temperature of the DHN.

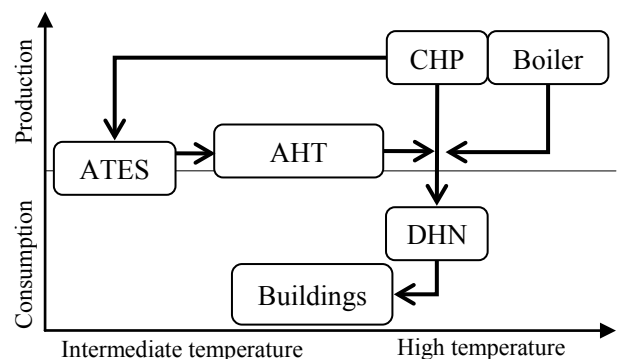


Figure 3. Energy concept schema with main thermal energy flows.

The system works as sketched in Figure 1 and Figure 3. The CHP produces a constant amount of energy to

cover the electric base load of the district. Its thermal energy is used to supply the buildings. In case there is a surplus of thermal energy, i.e., the amount of produced thermal energy is higher than the current demand of the DHN, this energy is used to charge the aquifer. If the thermal energy produced by the CHP does not cover the heat demand of the buildings, then, if available, thermal energy will be drawn from the aquifer. This heat energy then powers the AHT. In case the thus supplied thermal energy is still insufficient, either because of the limited capacity of the AHT or the depletion of the ATES (production temperature below minimum usable temperature), a boiler covers the remaining demand gap.

2 Modeling

The university campus Berlin-Charlottenburg is used as a case study for developing the modeling approach. In a first step, a model with simplified components and boundary conditions was developed in order to investigate the interaction of the sub-models.

The system model is assembled from component models from the MSL - Modelica Standard Library (Modelica-Association 2014), the Annex60 library (Wetter & Treeck 2014), the BuildingSystems library (Nytsch-Geusen 2014) and models specifically implemented for this project. The used models from the different libraries has fluid ports and input/output connectors as interfaces. Fluid ports are used to connect models mimicking the hydraulics of the real system while input and output connectors are mainly used for control signals.

The following subsections explain the overall system and its subsystems in more detail.

2.1 Overall System

Figure 4 shows the hydraulic configuration of the Modelica model, with dashed and solid lines representing heat and mass flows, respectively. At the

left and mid side of the diagram there is the thermal energy production system, composed by several production units, CHP, boiler and AHT, as well as per the seasonal thermal energy storage, ATES. At the very right side of the diagram there is the main consumption system, the building model, and its interface to the thermal energy production system, the DHN. The main components are either connected directly or via heat exchangers (HXs).

The control strategy of the model is explained below with help of Figure 4.

\dot{m}_{boiler} , the mass flow rate through the boiler and HX₃ is adjusted by a PI controller so that the DHN can extract the required energy in order to raise the supply mass flow rate, \dot{m}_{DHN} , to a certain set temperature. In this simulation it is set to a constant value of 90 °C.

The CHP model calculates, according to its inlet temperature and its constant heat production rate, the exact mass flow rate \dot{m}_{CHP} that can be heated up to a set temperature. In this simulation it is set to a constant value of 95 °C.

Taking into account that the boiler and CHP has the same set temperature, then, if the mass flow rate in the CHP \dot{m}_{CHP} is higher than the mass flow rate flowing through the boiler \dot{m}_{boiler} (used to heat up the DHN), more energy is produced by the CHP than requested by the DHN. In this case, the excess part of \dot{m}_{CHP} , which is not needed to heat the DHN ($\dot{m}_{\text{surplus}} = \dot{m}_{\text{CHP}} - \dot{m}_{\text{boiler}}$ and $\dot{m}_{\text{T2}} = 0$), is used to charge the aquifer ($\dot{m}_{\text{charge}} > 0$). In the opposite case the mass flow rate requested by the CHP is lower than the one requested to heat up the DHN ($\dot{m}_{\text{CHP}} < \dot{m}_{\text{boiler}}$), the mass flow rate leaving the heat exchanger of the DHN \dot{m}_{boiler} is split, one fraction goes to the CHP, \dot{m}_{CHP} , and the rest flows through the parallel branch, which is connected to the AHT ($\dot{m}_{\text{T2}} = \dot{m}_{\text{boiler}} - \dot{m}_{\text{CHP}}$ and $\dot{m}_{\text{surplus}} = 0$).

In this case, the AHT is switched on given that two conditions are fulfilled. A minimum mass flow rate

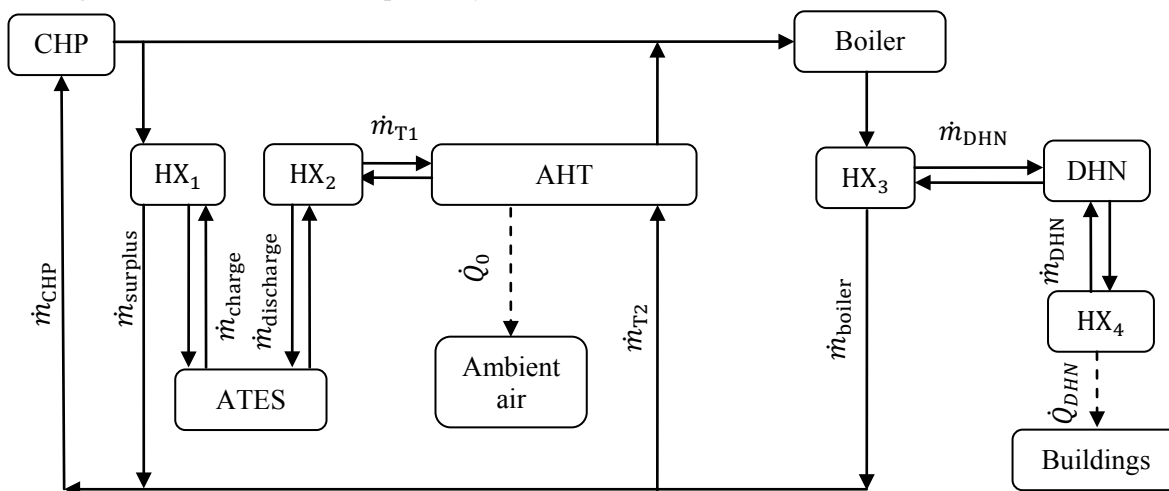


Figure 4. System schema of the Modelica model with main mass (solid) and thermal energy (dashed) flows.

\dot{m}_{T_2} and a minimum characteristic temperature difference $\Delta\Delta T$ (see 2.4). These conditions are not fulfilled when the ATES temperature is too low or when the ambient temperature limiting T_0 is too high. During the operation of the AHT energy is extracted from the ATES and used to drive the AHT ($\dot{m}_{\text{discharge}} > 0$ and $\dot{m}_{T_1} > 0$).

2.2 Combined Heat and Power Plant and Boiler

Boiler and CHP are represented by simple models, because the major interest of the simulation lies on ATES, AHT, DHN and buildings as thermal consumers.

The CHP works under stationary conditions and can be therefore considered as a constant heat source. The CHP is modeled with a prescribed heat flow. Assuming an electrical output of the CHP to cover the electric base load of 10 MW and a CHP ratio electricity/heat of 0.6 leads to a constant heat production rate of 15 MW.

The boiler's task is to ensure that the buildings' heat demand is fulfilled. We neglect any limitations and assume that this is always achieved. The boiler is modeled with a prescribed output temperature. The boiler's power is not limited and its output temperature is set to 95 °C.

2.3 Aquifer Thermal Energy Storage

The ATES model consists of two well models coupled by the produced/injected flow. This flow is heated or cooled in a heat exchanger which connects the ATES with the energy system. Both sides of the heat exchanger have equal mass flow rates. Well mass flow can be increased via a bypass in order to limit the injection temperature to $T_{\text{inj}}^{\text{max}}$. In the presented scenario we assume $T_{\text{inj}}^{\text{max}} = 60$ °C. In reality, well mass flow rate is limited by friction pressure losses in the well and in the aquifer with being a function of well diameter and rock permeability. However, as pressure loss is not considered here, well mass flow rate is limited by a prescribed value, representing the storage size together with the aquifer thickness.

The aquifer around the wells is modeled as two independent radially symmetric discs or rings with an inner radius r_{if} (interface well/aquifer) and an outer diameter r_{∞} . They are thermally insulated in the vertical direction. Thermodynamic equilibrium between rock and fluid, i.e., a common temperature is assumed. Eq. (1), the PDE¹ for convective-conductive transient radial heat flow in porous media (Bear & Bachmat 1990) written in polar coordinates provides the temperature $T(r, t)$ for a given Darcy flux, which is determined by the radial distribution of the mass flow entering/leaving the well \dot{m} as $u(r) = \frac{\dot{m}}{\rho^f 2\pi r H}$.

$$\rho c_p \frac{\partial T}{\partial t} = \frac{1}{r} \frac{\partial}{\partial r} k_{\text{eff}} r \frac{\partial T}{\partial r} - \rho^f u c_p^f \frac{\partial T}{\partial r} \quad (1)$$

H is the thickness of the aquifer. The effective thermal conductivity k_{eff} is calculated from the thermal conductivity of the rock k and the dispersion length D :

$$k_{\text{eff}} = k + D|u|c_p^f \rho^f \quad (2)$$

$\bar{\rho}$ is the bulk density, the average of fluid density ρ^f and rock density ρ^s , weighted with their respective volume fraction. The specific bulk heat capacity \bar{c}_p is calculated likewise.

$$\bar{\rho} = P \cdot \rho^s + (1 - P) \cdot \rho^f \quad (3)$$

$$\bar{c}_p = P \cdot c_p^s + (1 - P) \cdot c_p^f \quad (4)$$

At the outer boundary temperature is kept at the domain's initial value T_{∞} :

$$T(r, t = 0) = T_{\infty} \quad (5)$$

$$T(r = r_{\infty}, t) = T_{\infty} \quad (6)$$

r_{∞} is set to a large value so that the boundary is outside of the thermally influenced region. To verify this condition eq. (7) is monitored by an assert statement:

$$\left. \frac{\partial T}{\partial r} \right|_{r_{\infty}} = 0 \quad (7)$$

At the inner boundary temperature is set to injection temperature during injection or else heat flow is set to zero.

$$\begin{aligned} T(r_{\text{if}}) &= T_{\text{inj}} \quad \text{during injection} \\ \left. \frac{\partial T}{\partial r} \right|_{r_{\text{if}}} &= 0 \quad \text{else} \end{aligned} \quad (8)$$

The PDE is solved on the spatially discretized 1D domain using the FVM method (see Appendix).

The aquifer model has been successfully validated against a 3D FEM model simulated in COMSOL.

The aquifer model parameters are given in Table 1. Water properties are provided by the MSL.

Table 1. Aquifer model parameters

H	thickness	10 m
	initial temperature	20 °C
ρ_{rock}	rock density	2650 kg/m ³
k	rock thermal conductivity	3 W/(m·K)
$c_{p,\text{rock}}$	rock specific heat capacity	800 J/(kg·K)
P	porosity	0.3
r_{if}	radius interface well/aquifer	0.1 m
r_{∞}	radius of outer boundary	200 m

2.4 Absorption Heat Transformer

The AHT is a thermally driven heat pump of type II, which splits a heat flow at intermediate temperature level in two heat flows at high and low temperature levels (see Figure 5). The gain of the AHT is the output heat flow \dot{Q}_2 at the high temperature level T_2 . The effort is the driving heat flow \dot{Q}_1 at the intermediate

¹ partial differential equation

temperature level T_1 (equation (13)). With this device it is possible to provide heat for heating (e.g. the district) at T_2 driven by heat from the ATES at T_1 without using additional energy.

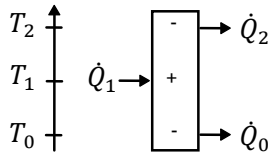


Figure 5. Black box scheme of an absorption heat transformer (Heat flows at different temperature levels).

The AHT is modeled following the steady-state approach of the characteristic equation. This approach was often applied and explained by absorption chiller (Albers et al. 2008; Puig-Arnabat et al. 2010). Cudok & Ziegler (2015) discuss the method of the characteristic equation for general apply to AHT and absorption heat pump (type I) and chiller.

The heat flows \dot{Q}_0 and \dot{Q}_2 are described by the characteristic temperature difference $\Delta\Delta T$ or respective the temperatures levels T_2 , T_1 and T_0 . The heat flow \dot{Q}_1 results from the energy balance eq. (12). To define a concrete AHT device the parameters, s_2 , s_0 , $\Delta\Delta T_{\min,2}$, $\Delta\Delta T_{\min,0}$ and B are needed.

$$\Delta\Delta T = B \cdot (T_1 - T_0) - (T_2 - T_1) \quad (9)$$

$$\dot{Q}_2 = s_2 \cdot (\Delta\Delta T - \Delta\Delta T_{\min,2}) \quad (10)$$

$$\dot{Q}_0 = s_0 \cdot (\Delta\Delta T - \Delta\Delta T_{\min,0}) \quad (11)$$

$$\dot{Q}_1 = \dot{Q}_0 + \dot{Q}_2 \quad (12)$$

$$\text{COP} = \frac{\dot{Q}_2}{\dot{Q}_1} \quad (13)$$

The AHT switches off when the characteristic temperature difference $\Delta\Delta T$ drops below a minimum value chosen to prevent the model from leaving the valid operating range, here this minimum value is set to 3 K, or when the mass flow rate \dot{m}_{AHT} is lower than a minimum value. This minimum mass flow rate is determined by the available gain heat flow \dot{Q}_2 of the AHT (equation (10)) and the assumed condition of a maximal temperature rise, which value set to 15 K is added to ensure normal operating conditions for the AHT. To switch off the device it is bypassed.

The AHT is controlled by the mean temperature at the low temperature level T_0 (equation (9)), which is variable but limited by the current ambient temperature.

For the system simulation the AHT is parameterized in such a way that it has a nominal power of 2 MW under the following operating conditions; $T_2 = 75^\circ\text{C}$, $T_1 = 60^\circ\text{C}$, $T_0 = 5^\circ\text{C}$.

2.5 Buildings

For use in the system model the approx. 50 buildings of the university campus Berlin-Charlottenburg with a total of 434020 m² heated floor space are aggregated into one substitute building model to ensure fast computation times of this component (see Figure 6).

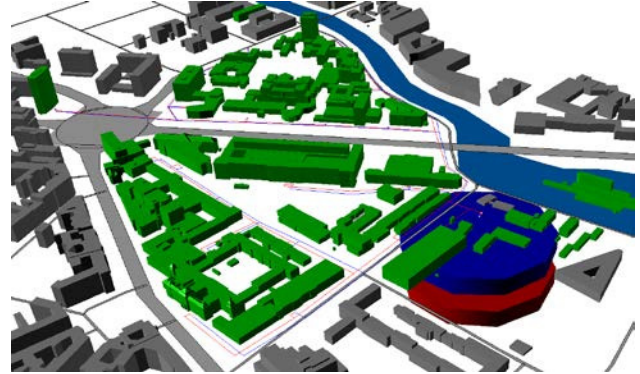


Figure 6. Building stock of the university campus Berlin-Charlottenburg (green), which is aggregated in one substitute building model. The blue and red cylinder indicate a possible location for the ATES.

The aggregation is done via a parameter identification method that uses optimization strategies, described in (Inderfurth et al. 2015). This method takes measured heat consumption data of all considered buildings into account. By repeated simulation of the substitute building model and comparing simulated and measured consumption important parameters like thermal capacities, thermal transmittances, window areas, etc. are gradually fitted to the building model. This process ensures that the substitute model together with its fitted parameter set accurately represents the thermal characteristic of the city district. The underlying low-order building model consists of four thermal capacities (internal capacities, external wall, base plate and room air), four window models to account for solar heat gains, several wall constructions, air change with the building's environment and an internal heat source accounting for internal heat gains through electrical consumption. All component models are provided by the BuildingSystems library for building performance simulation (Nytsch-Geusen 2014).

The low-order building model calculates the ideal, instantaneous heating demand of the aggregated buildings by calculating heat gains and losses to the environment through thermal conduction, air change as well as short and long wave radiation. The maximum heating power of the implemented ideal heater is reasonably limited to 30 MW. A previous study with measured heat consumption and weather data concludes that the maximum instantaneous heating load of the campus is around 23 MW.

2.6 District Heating Network

The district heating network's design is based on the information collected about the buildings' contracted heating power, the buildings' installations and their current operating temperatures. The DHN has been designed following a standard methodology (Krimmling 2011). The proposed district network has a tree structure and is simplified following the aggregation method described in Larsen et al. (2001). This method allows calculating the global heat loss and energy transport delay but not the pressure loss. The method has shown good results even after taking strong assumptions and extreme simplification of the distribution networks (Larsen et al. 2001, 2003). One of the strongest assumptions used by the aggregation method is that the ratio between mass flow rates in a pipe-branch is constant and proportional to the nominal power of the downstream consumer's nominal mass flow rate. Under this assumption the DHN is simplified step by step and the consumption points with it. Thus, once the DHN is reduced to one pipe/substation there is no need to redistribute the total energy demand of the city district between the different consumption points. In the present model the number of pipes was reduced from 75 to 1.

The thermal pipe model itself consists of a water volume with a lumped heat capacity from the pipe's metal and a heat transfer model that computes the heat transfer between pipe and surroundings. The heat transfer model for district heating pipes is described with equations (14) and (15) (Bøhm 2000).

$$\dot{Q}_s = (U_1 - U_2)(T_s - T_g) + U_2(T_s - T_r) \quad (14)$$

$$\dot{Q}_r = (U_1 - U_2)(T_r - T_g) - U_2(T_s - T_r) \quad (15)$$

T_s , T_g and T_r stand for supply, undisturbed ground and return temperature respectively. U_1 is the overall length-specific coefficient of heat transfer (HTC) between the pipe and the environment. U_2 is the HTC between the supply and return pipe. They are described by equation (16) and (17).

$$U_1 = \frac{R_g + R_i}{(R_g + R_i)^2 - R_m^2} \quad (16)$$

$$U_2 = \frac{R_m}{(R_g + R_i)^2 - R_m^2} \quad (17)$$

The values of the ground thermal resistance R_g , the thermal resistance of insulation and casing R_i , and the thermal resistance associated to the interaction between the two pipes R_m , can be found in Bøhm (2000).

For the calculation of heat flows it is assumed that the heat exchange between supply and return pipes is very low, $U_1 \gg U_2$. Hence, the right terms in equation (14) and (15) are neglected, leading to a simpler equation. Furthermore the heat conductivity of the ground and isolation material are kept constant.

The undisturbed ground temperature, T_g , at a depth z at time t is calculated with the equation (18) (Kusuda & Achenbach 1965). Here \bar{T} is the mean temperature of the ground surface for the entire year, A the annual amplitude of the ground surface temperature, t_0 a time shift that corresponds to the time at which the surface ground temperature has its minimum value, $\bar{T} - A$, and α is the thermal diffusivity of the soil. Equation (18) assumes a sinusoidal surface temperature with an oscillation period of one year. For the simulation, the time shift t_0 is set to 180 days and the equation is evaluated at the center of the pipe, $z = 1$ m.

$$T_g(z, t) = \bar{T} + Ae^{\left(-z\sqrt{\frac{\pi}{365d\cdot\alpha}}\right)} \cos\left(\frac{2\pi}{365d}\left(t-t_0-\frac{z}{2}\sqrt{\frac{365d}{\pi\alpha}}\right)\right) \quad (18)$$

The pipe model is parameterized with a length 781 m and an overall length-specific HTC $U_1 = 0.42$ W/mK. The pipe is discretized in five elements.

3 Simulation

The modeling and simulations of the proposed system are mainly conducted to study the interaction between subsystems and towards a correct dimensioning of the different subsystems.

3.1 System under simplified weather conditions

For the simulation, in order to facilitate understanding of the system's behavior, simplified boundary conditions are used to create a seasonal behavior free of daily variations. Hence, synthetic weather data was defined with a sinus curve with a period of one year for the ambient temperature and an amplitude of 25°C. Solar irradiation was neglected.

The simulation has been run for a period of eight years in Dymola using the solver DASSL with a tolerance value of 10^{-5} .

Like Figure 1, **Figure 7** shows the fluctuating thermal energy demand (negative values) and production (positive values) of the different subsystems.

At the beginning of the simulation the ATEs has not been charged yet. Hence the AHT cannot be switched on and the district's thermal energy demand is covered by the CHP and the boiler. Later, when the thermal energy demand falls below CHP's thermal energy production, the surplus energy is used to charge the ATEs. Thus, in the next heating period there is thermal energy stored in the aquifer, which can be used via the AHT until the fluid temperature from the aquifer reaches a certain minimum value at which the characteristic temperature difference ΔT is smaller than its selected minimum value (see 2.4).

The cycle repeats in the following years with the AHT operating period as well as the amount of thermal energy produced extending as shown in Table 2.

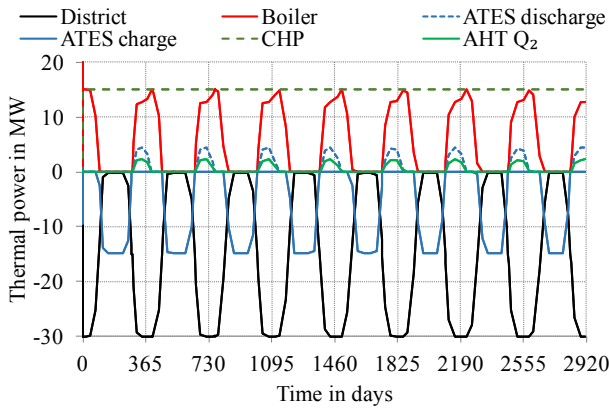


Figure 7. Thermal energy produced (positive values) and used by the different subsystems: Absorption heat transformer (AHT), combined heat and power plant (CHP), boiler, aquifer thermal energy storage (ATES) and district.

Table 2. AHT operation time in hours and energy produced in MWh after each seasonal discharge cycle.

Discharge cycle	AHT's operation time in hours	AHT's Energy produced in MWh
1	2477	4511
2	2554	4764
3	2568	4770
4	2580	4775
5	2588	4780
6	2596	4784
7	2602	4788

The AHT operation phase extends because the amount of thermal energy in the warm well after each charge cycle increases during the warm-up phase. The warm well injection temperature is limited to $T_{inj}^{max} = 60\text{ }^{\circ}\text{C}$ and the initial aquifer temperature is $20\text{ }^{\circ}\text{C}$, while the AHT extracts heat with a rather small temperature difference of 15 K. Hence, the stored heat is not recovered completely and the cold well injection temperature is much higher than the initial aquifer temperature until the cold side is warmed up. It means that during the warm-up phase, the extracted fluid from the cold well and reinjected through the warm well into the aquifer undergoes a smaller temperature. Hence more mass is heated up to the injection temperature than the previous charge cycle. Thus the thermal energy stored into the warm well increases year after year until the warm-up phase is over.

Figure 8 shows the ratio of recovered heat Q_{out} to injected heat into the aquifer Q_{in} for different injection temperatures T_{inj} in the warm well, calculated as heat recovery factor (HRF) by the following equation (Kranz & Bartels 2009):

$$HRF(\tau_i) = \frac{Q_{out}}{Q_{in}} = \frac{\int_0^{\tau_i} \min(0, \dot{m}_w) \cdot (h_w - h_c) dt}{\int_0^{\tau_i} \max(0, \dot{m}_w) \cdot (h_w - h_c) dt} \quad (19)$$

\dot{m}_w is the mass flow rate into the warm well, $h_{w/c}$ is the specific fluid enthalpy at the warm/cold well head. τ_i is the end time of the i -th discharge cycle.

The HRF is between 0 and 1 if the warm well temperature is always above cold well temperature and the cold well injection temperature is above the initial aquifer temperature.

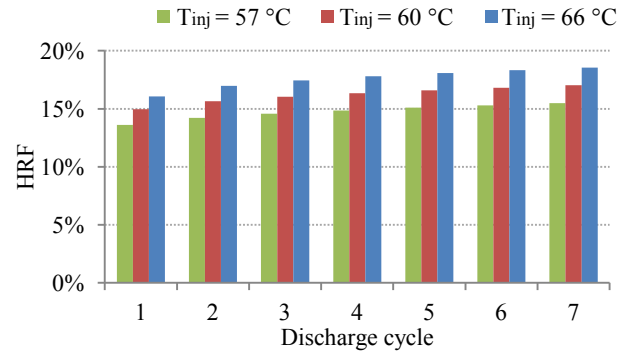


Figure 8. Aquifer heat recovery factor (HRF) for different injection temperatures T_{inj} in the warm well.

As explained above, the HRF increases over time because the cold side of the aquifer is warmed up in the first cycles. This is a common behavior of an ATEs system. Furthermore, a higher injection temperature yields higher recovery factors.

3.2 System under real weather conditions

Simulating the system with synthetic weather boundary conditions, as described above, is vital for the fundamental understanding of the functionality of the described, complex system. However, to deduce meaningful results for real-world applications, the system model has to be simulated with actual weather input.

Usually real weather data used as input for simulations has larger gradients than synthetic, sinusoid weather data, as well as hourly spacing of samples, which requires interpolation.

Simulations with actual, measured weather data and the described system model have been performed successfully. Results will be presented in subsequent publications.

4 Discussion

The simulation results with simplified boundary conditions show that the stored energy cannot be used completely to drive the AHT, which yields low values of HRF. It is caused by a relatively high minimum value of the driving temperature T_1 over which the AHT can operate. This minimum temperature is obtained from equation (9) evaluated at $\Delta\Delta T = \Delta\Delta T_{min} = 3\text{ }^{\circ}\text{C}$ and specific values of T_0 (see Figure 9).

Figure 9 shows that in order to use low values of the AHT's driving temperature T_1 , temperatures T_0 and T_2 must be low as well. T_0 is related to the ambient temperature (see chapter 2.4), in this regard it is

interesting to favour the utilization of the AHT during the coldest months and at night. The minimum value T_2 is determined by the temperature range of the DHN. In this regard, low temperature DHN are favourable. Furthermore, the AHT should preheat the fluid before the CHP brings it to the temperature required by the DHN.

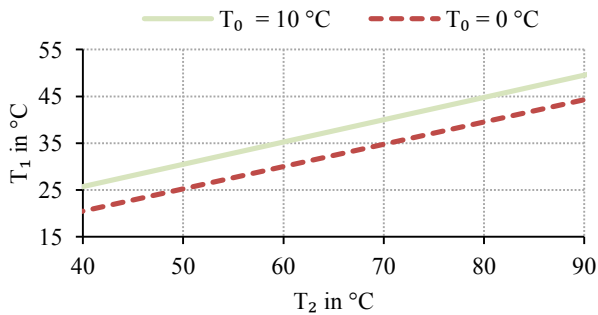


Figure 9. Minimum value of T_1 in dependency of T_2 for two different values of T_0 assuming $\Delta T_{\min} = 3$ °C and $B = 1.1$.

The simulation results also show that the cold side of the aquifer needs several cycles to reach stable conditions. In order to shorten this warm-up phase with low HRF, it would be interesting to use a low cold well injection temperature so that less energy is needed warming up the cold side of the aquifer. Alternatively, one could reduce the distance between cold and warm well (Kranz et al. 2015). Modeling the interaction of the two wells would require a more detailed 2D/3D model for the aquifer.

Furthermore, the results show how a higher injection temperature yields higher HRF. In this regard, it is important to properly assess which is the upper temperature limit for safe operation of the ATES. Furthermore, if the warm well production temperature is sufficiently high to be used for preheating the return temperature of the DHN, this direct heat transfer is recommended.

The results obtained with real weather data highlights the importance of a correct sizing of the CHP. An oversized CHP reduces and shorts the period of times in which the AHT can work, causing the system to recover just part of the stored energy in the aquifer, thus obtaining low HRF. On the other hand, too small CHP produces less surplus energy and can make the utilization of an ATES system a nonsense.

5 Conclusions and Future Work

Modelica proved to be a proper framework for the development of models for the different subsystems by different research group as well as for their subsequent integration in a more complex model thanks to the use of common interfaces. Furthermore, Modelica and DYMOLA proved suitable for simulation of complex energy systems.

The simulation results show the expected thermal behavior of the different subsystems.

Different factors limiting the efficiency of the ATES system were pointed out. In this regard, a possibility to increase the HRF of the ATES was proposed.

Furthermore, the benefit of seasonal storage is shown as in the case study heat is recovered from the ATES which would otherwise have been lost as waste heat. Assessing the low values of the calculated HRF one has to allow for the simplified system design with simple boundary conditions, which has a lot of potential for improvement.

We are working towards the modeling of a more sophisticated system, including cooling demand, heat demand at different temperature levels and its pertinent distribution networks. Furthermore, we envisage increasing the level of modeling detail by adding new models for solar production, absorption heat pump type I and an absorption chiller, as well as a 2D/3D aquifer model and more sophisticated control strategy.

Acknowledgements

The research described in this article is conducted within the research project ATES: Aquifer Thermal Energy Storage (Effizienz und Betriebssicherheit von Energiesystemen mit saisonaler Energiespeicherung in Aquiferen für Stadtquartiere) funded by the Federal Ministry for Economic Affairs and Energy in Germany (BMWi 03ESP409A/B/C).

Nomenclature

A	Annual temperature amplitude	K
AHT	Absorption Heat Transformer	-
ATES	Aquifer Thermal Energy Storage	-
B	Duehring factor	-
CHP	Combined Heat and Power	-
COP	Coefficient Of Performance	-
DHN	District Heating Network	-
D	Dispersion length	m
H	Thickness of the aquifer	m
HRF	Heat Recovery Factor	-
HTC	Heat Transfer Coefficient	-
HX	Heat exchanger	-
Q_x	Heat	kWh
\dot{Q}_x	Heat flow rate	kW
P	Porosity	$\frac{m^3}{m^3}$
R_g	Ground's thermal resistance	$\frac{mK}{W}$
R_i	Insulation's thermal resistance	$\frac{mK}{W}$
R_m	Thermal resistance between pipes	$\frac{mK}{W}$
\bar{T}	Mean annual temperature	K
T	Temperature	K

U_1	Overall length-specific HTC between pipe and environment	$\frac{W}{mK}$
U_2	Overall length-specific HTC between supply and return pipe	$\frac{W}{mK}$
$c_p^f/c_p^s/\bar{c}_p$	Specific heat capacity of fluid/rock/bulk	$\frac{J}{kgK}$
h_w/c	Specific fluid enthalpy at the warm/cold well head	$\frac{J}{kgK}$
r_{if}	Radius of interface well/aquifer	m
r_∞	Radius of outer aquifer boundary	m
s	Slope parameter	$\frac{W}{K}$
t	Time	s
t_0	Time shift	d
k	Rock thermal conductivity	$\frac{W}{mK}$
k_{eff}	Effective thermal conductivity	$\frac{W}{mK}$
\dot{m}	Mass flow rate into the well	$\frac{kg}{s}$
u	Darcy flux/velocity	$\frac{m}{s}$
z	Depth	m
α	Thermal diffusivity	$\frac{m^2}{s}$
$\Delta\Delta T$	Characteristic temperature difference	K
$\Delta\Delta T_{min,X}$	Loss parameter	K
$\rho^f/\rho^s/\bar{\rho}$	Density of fluid/rock/bulk	$\frac{kg}{m^3}$
τ_i	End time of i-th discharge cycle	s

Appendix – Discretized heat transfer equation

The equation by Bear & Bachmat (1990) was formulated in polar coordinates and reduced to the radial component to yield eq. (1), which describes the conductive-convective transient radial heat flow in porous media:

$$\rho c_p \frac{\partial T}{\partial t} = \frac{1}{r} \frac{\partial}{\partial r} k_{eff} r \frac{\partial T}{\partial r} - \rho^f u c_p^f \frac{\partial T}{\partial r}. \quad (1)$$

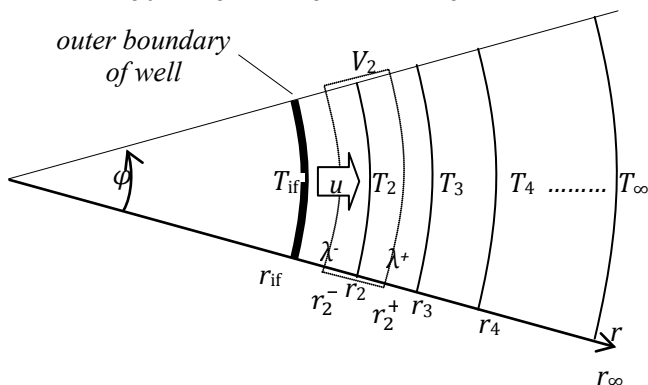


Figure 10. Radially equidistant discretization of aquifer

In order to solve eq. (1) numerically, the radially symmetric aquifer was subdivided in equidistant sections as shown in Figure 10 and the equation was discretized as follows.

First, it is integrated over the ring segment V assuming constant ρ , c_p and ρ^f :

$$\int_V \bar{\rho} \bar{c}_p \frac{\partial T_i}{\partial t} dV = \int_V \left(\frac{1}{r} \frac{\partial}{\partial r} k_{eff} r \frac{\partial T}{\partial r} - \rho^f u c_p^f \frac{\partial T}{\partial r} \right) r d\phi dr dz$$

$$\bar{\rho} V \bar{c}_p \frac{\partial T_i}{\partial t} = 2\pi H \int_{r_-}^{r_+} \left(\frac{\partial}{\partial r} k_{eff} r \frac{\partial T}{\partial r} - \rho^f u r c_p^f \frac{\partial T}{\partial r} \right) dr.$$

By substituting $\rho^f u H 2\pi r = \dot{m}$ one obtains

$$\bar{\rho} V \bar{c}_p \frac{\partial T_i}{\partial t} = \left(2\pi H k_{eff} r \frac{\partial T}{\partial r} - \dot{m} c_p T \right) \Big|_{r_-}^{r_+}.$$

Finally, the discretized equation is obtained using the symmetric first order difference quotient and an upstream velocity scheme for the convective term:

$$\bar{\rho} V \bar{c}_p \frac{\partial T_i}{\partial t} = 2\pi H k_{eff} \left(r_+ \frac{T_{i+1} - T_i}{r_{i+1} - r_i} - r_- \frac{T_i - T_{i-1}}{r_i - r_{i-1}} \right) - \dot{m} c_p \begin{cases} T_{i+1} - T_i & \text{if } u < 0 \\ T_i - T_{i-1} & \text{if } u > 0 \end{cases}$$

where $V = 2\pi r H$, k_{eff} and u are functions of r and hence segment specific (index i has been omitted for brevity). $\bar{\rho}$, \bar{c}_p and k may be varying or constant.

References

- Albers, J., Kuehn, A., Petersen, S., & Ziegler, F. Control of absorption chillers by insight: the characteristic equation. Krakau. 2008
- Bear, J., & Bachmat, Y. *Introduction to modeling of transport phenomena in porous media*, Vol. 4. Springer Science & Business Media. 1990
- Bøhm, B. On transient heat losses from buried district heating pipes. *International journal of energy research*. 2000
- Cudok, F., & Ziegler, F. Absorption heat converter and the characteristic equation method. *International Congress of Refrigeration*. Yokohama, Japan. 2015
- Dinçer, I., & Rosen, M. A. *Thermal Energy Storage: Systems and Applications*. John Wiley & Sons, Ltd. 2002
- Inderfurth, A., Nytsch-Geusen, C., & Ribas Tugores, C. Parameter identification for low-order building models using optimization strategies. *14th international Conference of the Building Performance Simulation Association (IBPSA)*. Hyderabad, India. 2015
- Kranz, S., & Bartels, J. Simulation and data based identification of parameters affecting seasonal ATEs efficiency. *Effstock 2009*, pp. 1–8. Stockholm, Sweden. 2009
- Kranz, S., Bloecher, G., & Saadat, A. Improving Aquifer Thermal Energy Storage Efficiency. *World Geothermal Congress*. 2015
- Krimmling, J. *Energieeffiziente Nahwärmesysteme*. Fraunhofer IRB. 2011

- Kusuda, T., & Achenbach, P. R. *Earth temperature and thermal diffusivity at selected stations in the united states*. National Bureau of Standards Gaithersburg MD. 1965
- Larsen, H. V., Bøhm, B., & Wigbels, M. A comparison of aggregated models for simulation and operational optimisation of district heating networks. *Energy Conversion & Management*. 2003
- Larsen, H. V., Pålsson, H., Bøhm, B., & Ravn, H. F. Aggregated Dynamic simulation model of district heating networks. *Energy Conversion & Management*. 2001
- Modelica-Association. Modelica Standard Library. 2014
- Nytsch-Geusen, C. Modelica Library Building Systems. 2014
- Puig-Arnabat, M., López-Villada, J., Bruno, J. C., & Coronas, A. Analysis and parameter identification for characteristic equations of single- and double-effect absorption chillers by means of multivariable regression. *International Journal of Refrigeration*, 33/1: 70–78. 2010 . DOI: <http://dx.doi.org/10.1016/j.ijrefrig.2009.08.005>
- Sanner, B., Kabus, F., Seibt, P., & Bartels, J. Underground Thermal Energy Storage for the German Parliament in Berlin, System Concept and Operational Experiences. *World Geothermal Congress 2005*. Antalya, Turkey. 2005
- Schmidt, T., Kabus, F., & Müller-Steinhagen, H. The Central Solar Heat Plant with Aquifer Thermal Energy Store in Rostock, Germany. *TERRASTOCK*. Stuttgart, Germany. 2000
- Wetter, M., & Treeck, C. van. IEA EBC Annex 60, New generation computational tools for building and community energy systems based on the Modelica and Functional Mockup Interface standards. 2014

Energy-Efficient Design of a Research Greenhouse with Modelica

Dipl.-Ing. Torsten Schwan¹ Dipl.-Ing. René Unger¹ B.A. Jörg Pipiorke²

¹EA Systems Dresden GmbH, Germany, {torsten.schwan, rene.unger}@ea-energie.de

²ITI GmbH, Germany, pipiorke@itisim.com

Abstract

Greenhouses, especially for research applications, have high requirements on indoor climate control. The technical systems for heating, cooling, and moistening are more complex than in typical dwelling houses or office blocks and are highly dependent on local weather conditions. Increasing the energy efficiency and integrating renewable power into these systems is a sophisticated engineering task which requires extensive investigation.

This paper describes a combined approach to model and simulate building operation and HVAC system behavior of a research greenhouse with Modelica. This includes the presentation of some important modelling paradigms as well as system concept validation with some interesting simulation results.

Keywords: Green Building, Building Simulation, Solar Cooling, Greenhouse Design

1 Introduction

Greenhouses are used to cultivate a great variety of plants from all over the world during the whole year. Therefore indoor temperature and humidity have to be controlled at a certain plant-specific level regardless of environmental conditions. Furthermore, the cultivation of plants requires a sufficient amount of light. This way, greenhouses are built with excessive glass surfaces to optimally use solar radiation for the required lighting. However, buildings with large glass surfaces often overheat especially during summer time. To ensure a sufficient indoor temperature level even in times of high solar input and without losing humidity, this building type requires huge amounts of cooling energy which has to be provided by the building internal HVAC (i.e. Heating, Ventilation, Air Conditioning) system.

An optimal HVAC system design for greenhouses requires extensive analysis. Besides building physics (e.g. heat losses through walls and windows, ventilation) required plant-specific indoor temperature and humidity as well as solar radiation level highly affect heating and cooling system requirements. Furthermore, increasing energy prizes and today's demands for environmental protection require alternative system solutions. To reduce running costs as well as ecological footprint a suitable combination

of local renewable energy production, highly efficient building materials, and intelligent control algorithms is needed.

A multi-domain system simulation helps to consider all these aspects in one evaluation and optimization framework. This paper describes how SimulationX and the Modelica-based Green Building library were used to analyze, to evaluate, and to validate an innovative approach of greenhouse HVAC system design. Because main overall energy consumption of a greenhouse is used for cooling, the presented work mainly considers cooling energy system design.

The key concept is the usage of solar heat for indoor cooling, since the cooling system is mostly needed when it is sunny. Therefore, an absorption cooling machine supplied by two different types of solar thermal collectors and district heating is used to generate the required cooling energy.

The simulation approach is basically divided into two parts. The initial simulation of building physics and operation is followed by an accurate simulation of HVAC system behavior including optimized system control algorithms.

This paper gives an overview about some interesting aspects of building operation and HVAC system modeling. It presents a short description of the developed greenhouse building physics model including all inner loads, like assimilation lighting and humidification, as well as required control algorithms for lighting, shading and temperature control. Special emphasis is put on the identification and modelling of suitable parameters representing plant growth which is necessary to model indoor humidity behavior.

Furthermore, the contribution presents how to model the designed HVAC system with Green Building library components. Additionally necessary extensions of library components are presented. Finally, the paper compares some interesting results regarding parameter variation and adapted system configurations.

2 Building and HVAC System Concept

The planned greenhouse will be built in the city center of Leipzig, a major city in Eastern Germany. As a center of biological research, scientists and students of University of Leipzig will use it to identify and evaluate effects of global warming on indigenous



Figure 1: Floor plan of greenhouse section and social building (GEFOMA, 2015)

vegetation, and to perform further research-relevant experiments.

The planned building basically consists of a solid social building with basement and first floor as well as a glass-covered greenhouse section with an overall basic area of about 1.000 m². The greenhouse is divided into twelve cubicles to cultivate twelve different plant species at the same time. Temperature and humidity control thus require individual settings in each cubicle.

Planner's basic idea was to reduce the overall ecological footprint about 50% in comparison to a conventional greenhouse concept. This way, two different types of the building had to be compared regarding heat, cold and electricity demand.

Both building concepts, reference and alternative, mainly differ regarding greenhouse section

construction as well as planned HVAC system. Basically, the alternative greenhouse section construction substitutes single with insulating glazing. Furthermore, each glass surface in each cubicle can be shaded individually by a two-part shading system.

Temperature and humidity control is supported by a mechanical ventilation system using automatic top-hung windows in the roof ridge.

The planned HVAC system is divided into heating and cooling systems. Instead of a conventional oil-fired condensing boiler the alternative building uses locally available district heating as heat supply. This is a comparatively simple and energy-efficient but not that innovative solution.

Real innovation is planned regarding cooling system design. The reference building uses a chilled water unit cascade with dry chillers as cold supply. Opposite to

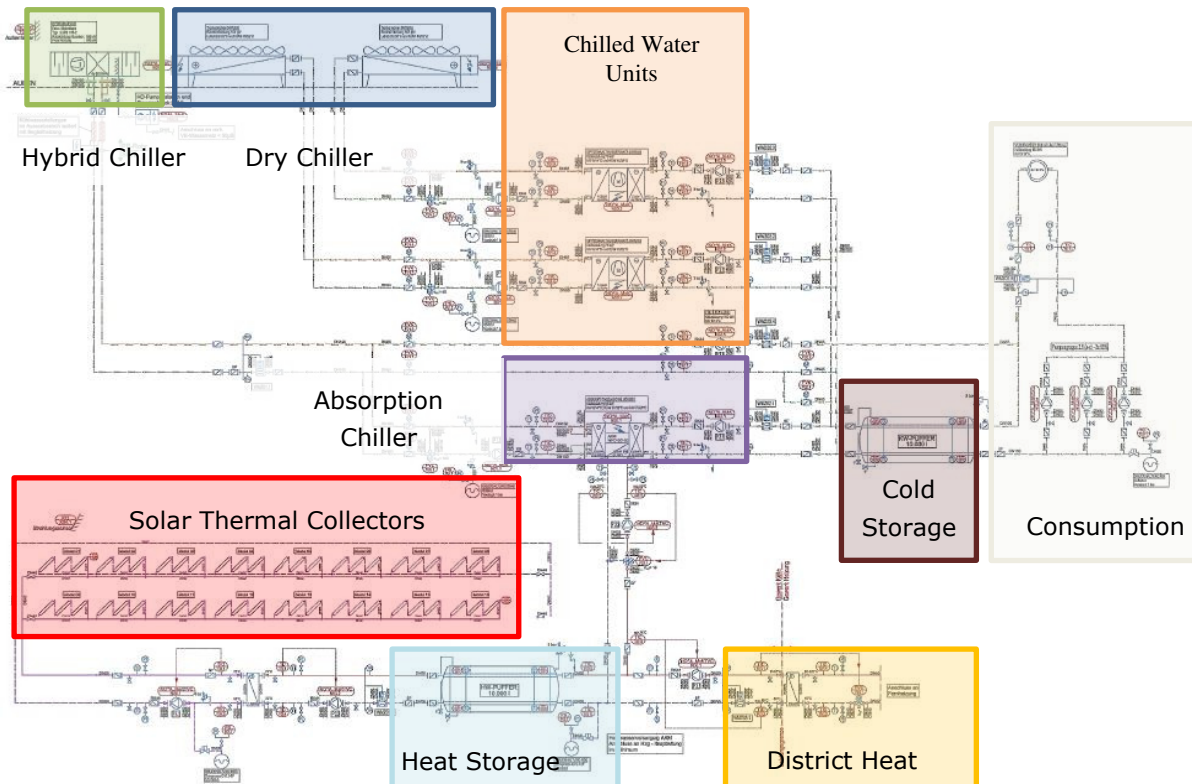


Figure 2: HVAC system concept (GEFOMA, 2015)

that, the alternative cooling system is based on an absorption chiller. This uses solar thermal collectors for heat source with district heating as backup. Furthermore, a small cascade of chilled water units provides peak cooling power in times of higher cooling demand.

Because of low required recooling temperatures (maximum 32°C even at ambient temperatures of 35°C) the absorption chiller uses a hybrid cooler as heat exchanger. Hybrid chillers provide lower recooling temperatures by additional cooling with latent heat of water evaporation.

As an extended research topic two different types of solar thermal collectors, direct flow and heat pipe, are integrated in the presented HVAC system concept. The collector surface area is evenly distributed between the two systems. Both collector systems have advantages and disadvantages at different operation points. This way, simulated solar gains will be compared to later measurements in a subsequent monitoring campaign after building completion.

Furthermore, greenhouses often require cooling energy even when ambient temperature is comparatively low, e.g. sunny winter days. In this case, dry and hybrid chillers can directly provide cooling energy without further use of electricity (Chilled Water Units) or heat (absorption chiller), i.e. free cooling. Because hybrid chillers produce lower recooling temperatures at same ambient temperature level, presented HVAC system concept prefers this system to provide free cooling. This way, free cooling is possible

to 10°C ambient temperature and 55% relative humidity.

3 Building Operation

Greenhouses widely differ from normal buildings. Besides excessive glass surfaces and extended control algorithms for window shading, heating and ventilation the cultivation of different plant species requires specific levels of indoor temperature and humidity. This way, building simulation has to put special emphasis on indoor climate. The calculation of inner heat gains by persons or electric components is not suitable anymore. Furthermore, greenhouses mainly require cooling energy. Cooling load calculation is a complex task by itself (VDI, 2007).

Basically, characteristics of cultivated plant species have major influences on resulting building behavior. They are mainly responsible for all relevant hygrothermal requirements in each cubicle, like indoor temperature, lighting, and humidity.

To include these major influencing characteristics into the simulation, a set of suitable plant characteristics have to be identified. The basic idea is to find one or two plant species with an average behavior of plants based on the following parameters:

- Indoor temperature (day and night),
- Relative humidity,
- Light intensity and illumination duration,
- Duration of measurement, and
- Plant transpiration.

These characteristics were scrutinized together with the specialists of University of Leipzig, the desired building operator. This way, two different kinds of parameter sets could be identified:

1. Global climate change on root characteristics:
 - a. 15°C/18°C by day, 8°C by night
 - b. 50% relative humidity
 - c. 1000 lx light intensity during 16h per day
 - d. 4 months measurement
2. Underground and aboveground germination:
 - a. 22°C by day, 15°C by night
 - b. 60% relative humidity
 - c. 10.000 lx light intensity during 16h per day
 - d. 3 months measurement

These characteristics adequately describe building requirements regarding lighting and indoor temperature. Apparently, the University of Leipzig will mainly use the greenhouse to cultivate indigenous plant species. That way, cooling energy demand will be comparatively high during summer time.

Together with indoor temperature, the relative humidity describes the hygrothermal reference behavior in each cubicle. However, the definition of a desired humidity set point is not sufficient to define relevant requirements to ventilation and humidification system. The transpiration of plants as a major influence on moisture is missing.

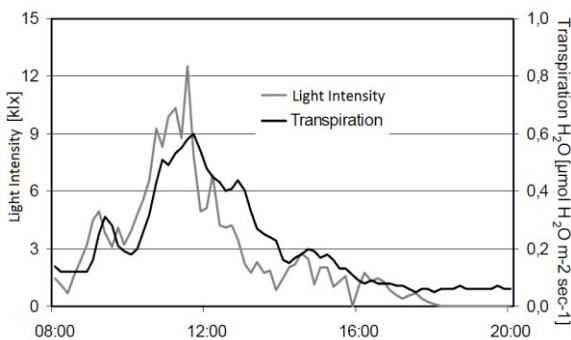


Figure 3: Correlation between light intensity and transpiration for a specific plant species (Bertram et. al., 2004)

To model the transpiration of plants the simple correlation between light intensity and transpiration was used (c.f. Figure 3). This way, values between 0.028 and $0.4378 \frac{mg}{m^2s} / \frac{W}{m^2}$ could be identified for different relevant plant species as additional model input parameters.

4 Green Building Library

The building and HVAC system were modeled using SimulationX and the Green Building library. This library was developed by EA Systems as a versatile simulation environment for renewable energy

systems and energy management design (c.f. Schwan et. al., 2012). 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. 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). The modeling focus lies on the interactive behavior of different energy system components with varying complexity in the context of building energy supply, either thermal or electrical (i.e. electrical systems modeled using RMS values). 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).

With its specific focus on a wide range of energy system components, Green Building library significantly distinguish from other well-arranged Modelica Building libraries, like *Modelica Buildings* library (Wetter, 2009) which are more intended to accurately model building physics behavior. This way, Green Building became the tool of choice because this project's main aim was to compare different types of HVAC systems with less accuracy requirements on thermal building behavior.

5 Building and HVAC System Modeling

The developed basic greenhouse building model consists of 38 thermal zones, each represented by one *Building Zone* model of the Green Building library. These thermal zones individually represent each room in the planned greenhouse, 7 zones for the basement and 16 zones for the first floor of the social building as well as 15 zones for the glass-covered greenhouse section (12 cubicles, 3 corridors).

However, the basic Green Building models are optimized for calculation speed, providing a simplified set of equations to describe simple thermal behavior in a building zone (e.g. solar gains, ventilation losses, transmission losses through walls and windows, etc.). Therefore a hygrothermal building zone model was developed based on this basic model component.

The new model includes lighting and shading control depending on solar radiation as well as ventilation control depending on indoor humidity conditions. Furthermore, physical effects representing plant-specific hygrothermal behavior were added as well:

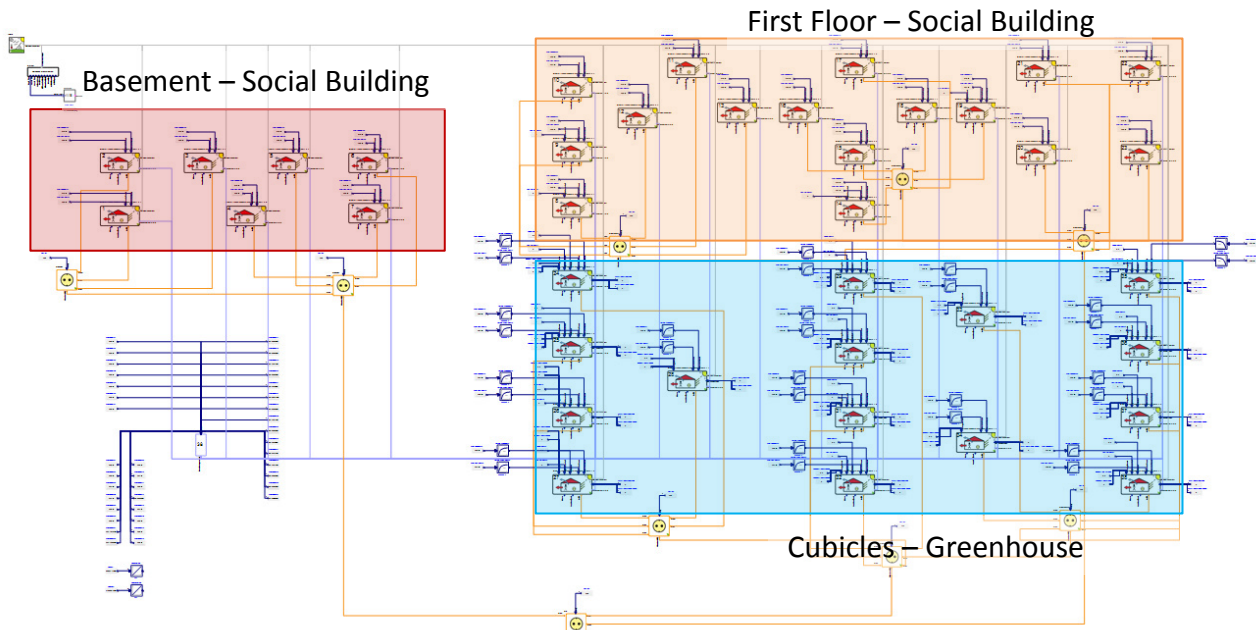


Figure 4: Building model of alternative greenhouse concept

- Evaporation heat of irrigation and humidification
- Condensation at cubicle walls and ceilings
- Transpiration of plants
- Ingress/input of moisture via persons

To compare both building concepts, the reference and the alternative building, two different building models were developed. Figure 4 shows the complete building model of the alternative building including all hygrothermal zones, weather conditions and grid as well as ventilation, lighting and shading control.

Each model additionally includes time schedules representing presence of research members in each zone (normally during the week between 8 a.m. and 6 p.m.) as well as electrical energy consumption via different relevant system components (e.g. refrigerators, server technology, autoclave, etc.).

The reference building model only uses additional lighting and ventilation control to keep required indoor temperature and humidity levels and to fulfill plant-specific lighting demand in each cubicle.

Furthermore, the alternative building model includes shading control for both planned energy screens. The first screen decreases solar irradiation in times of high indoor temperatures and second screen reduces heat losses via glass surfaces during night in times of low outdoor temperatures.

Both models only consider building physics, operation and hygrothermal behavior. There is no directly connected HVAC system model. However, the resulting time transient characteristics for heat, cold and electrical energy demand can further be used as input data sets for detailed HVAC systems simulation.

A direct link to the HVAC system models would be possible as well. However, the building model already

runs about 24h to simulate one year because of the required high accuracy regarding zoning, input data and analyzed physical effects (hygrothermal behavior). Because of relations between temperature control in each zone and available heat/cold from the HVAC system, linked models would increase simulation time to non-acceptable values.

This way, the HVAC system models are modeled separately. These models are again divided into heating and cooling system models. This is possible because building system simulation provides independent results for heat and cold demand. This approach further reduces simulation time.

Figure 5 shows the developed HVAC system model for cooling supply of the alternative building. Because of its structure, it can easily be compared to the original planner's system concept in Figure 2. All relevant components are individually modeled with existing or adapted Green Building model components.

However, Green Building library mainly considers heating system (e.g. Heat Pump, CHP) and electric components (e.g. batteries and eVehicles) as well as renewables (e.g. photovoltaics, solar thermal collectors). This way, some additional components based on Green Building modelling paradigms (Schwan et. al., 2011) had to be developed.

The main component of the cooling system concept is an absorption chiller. This component uses heat in internal chemical processes to produce cooling energy. This process is periodically reversed using external heat. Compared to conventional cooling systems, like chilled water units, the required amount of electricity can be significantly reduced. The absorption chiller model describes physical behavior based on a set of suitable pre-calculated operating points. This way full

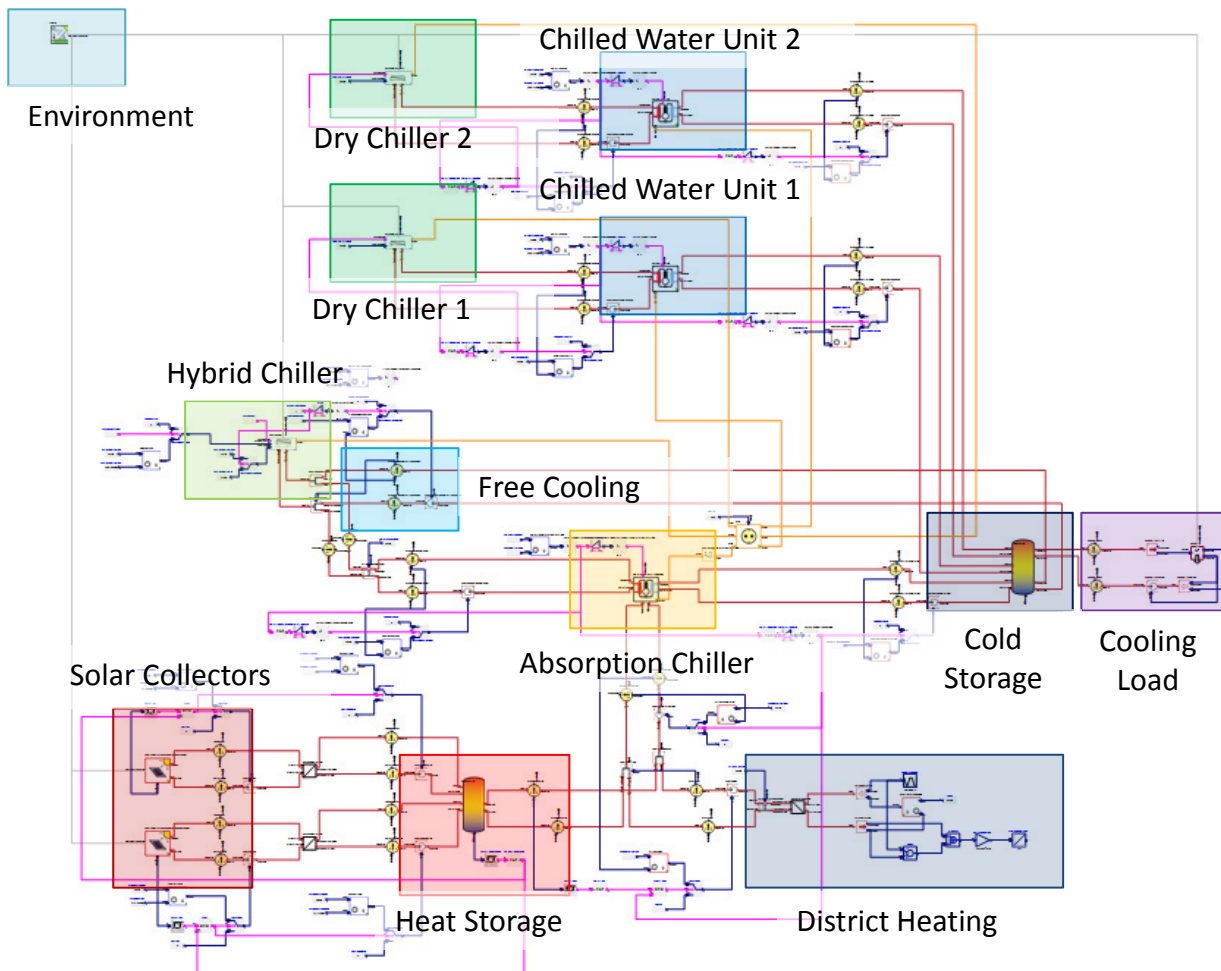


Figure 5: HVAC system model of alternative building (only cooling supply)

year simulations are possible, because the model runs very fast. Internal processes are transferred into a phenomenological behavior which is described as a black-box system.

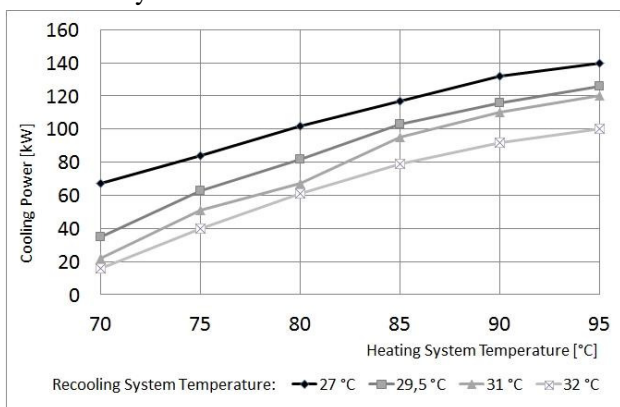


Figure 6: Sample characteristics of absorption chiller (Yazaki, 2015)

Basically, the new absorption chiller model has three interfaces to other system components, heating power input, recooling power output, and cooling power output. Each interface is divided into flow and return pipe with separate temperature levels but equal medium volume flow rate. The overall system behavior

is simulated depending on the temperature difference between each flow and return as well as externally controlled volume flows. Heating power input and cooling power output are defined by linearly interpolated Modelica *CombiTimeTables*. These tables are directly filled with OEM-specific system data, e.g. cooling power characteristics shown in Figure 6.

This way, system validation was quite simple because internal processes were neglected and alternatively substituted by already validated data sheet values. A short set of simulation runs with constant input data sets (e.g. flow and return temperatures) proved developed model approach and showed the right system behavior at different characteristic operating points.

Another new model represents the phenomenological behavior of chilled water units. These system components produce cooling energy via external cooling and a certain amount of electrical energy. They mainly work as inverted heat pumps. This way, the newly developed chilled water unit model basically uses already existing Green Building *Heat Pump* functionality. However, the corresponding controllers are redesigned to use cold instead of heat demand as reference control value. Basic model layout

and required tests are in accordance with previously presented absorption chiller model.

Both cooling system components require external coolers. Dry coolers only heat ambient air with heat from the cooling medium. Therefore, an external set of fans provide a specific amount of air to keep recooling temperature at a certain level.

Hybrid coolers additionally use evaporation heat of water to further cool down the cooling medium. They are especially needed in combination with absorption chillers because these components only work efficiently up to a maximum recooling temperature of 32°C. Otherwise, absorption chillers would switch off just in times of highest cooling demand.

The overall HVAC system model in Figure 5 includes both the physical system behavior of relevant cooling system components as well as associated control algorithms:

- Priority control of solar heat supply for the absorption chiller (solar collector temperature greater than 78°C)
- Flow temperature control via district heating supply (absorption chiller temperature greater 75°C and smaller 95°C)
- Priority control of the cooling system (base load via the absorption chiller, peak load via the chilled water units)
- Integration of free cooling in times of low outdoor temperature via hybrid chiller

6 Simulation Results

The presented HVAC system design was developed using existing theoretical and empirical knowledge of greenhouse layout. The basic idea of the alternative building concept was to reduce overall energy consumption for heating and cooling about 50% regarding the reference building concept.

However, solar aided cooling systems cannot be designed and calculated without adequate simulation models. The basic task of the system simulation was to validate both building concepts regarding intended energy saving potentials.

But before developed simulation models could be used to validate actual system concept, the models had to be validated regarding sufficient calculation results and relevant standards respectively tools. Therefore, the planner's original calculation results were used. The simulation results of greenhouse specific HORTEX 4.1 simulation tool (i.e. based on DIN EN ISO 13790 and DIN EN ISO 13789) showed an overall heating load of 287 kW for the reference and 196 kW for the alternative construction type. In comparison to that the simulation results of the developed Green Building models have been nearly at the same level (reference: 273 - 283 kW, alternative: 194 - 201 kW depending on chosen operation strategy, c.f. Table 1

and Figure 7). This way, developed models can be used for further validation analysis.

The first simulation results only describe the effects of different building operation strategies regarding overall heat and cold demand. Basically, three different operation strategies were analyzed with both building models (c.f. Figure 4):

- V1: All-year high cooling requirements
- V2: Reduced cooling requirements during summer time
- V3: All-year reduced cooling requirements (maximum indoor cubicle temperature 25°C)

All three strategies mainly refer to cooling requirements of the twelve cubicles. The presented temperature levels (c.f. section 3) are always set as minimum temperature level for heating system control. But cooling system control can be different.

Table 1: Comparison of heat and cold demand

Variant	Heat Demand [MWh/a]	Cold Demand [MWh/a]
V1 Reference	541.9	309.5
V1 Alternative savings	364.1 32.8 %	223.6 27.8 %
V2 Reference	532.7	140.0
V2 Alternative savings	355.0 33.4 %	106.8 23.7 %
V3 Reference	522.7	85.5
V3 Alternative savings	335.7 35.8 %	34.0 60.2 %

Table 1 shows a short comparison between different simulation results of heat and cold demand for all three operation strategies and both building types as well as resulting saving potentials.

Because all three strategies do not affect the heating system control strategy, the resulting heat demand for all strategies is about the same. Small differences are caused by the inner heat storage capacity of the building.

Furthermore, the simulation results show that modern greenhouse glazing as well as extended shading and lighting control algorithms are not sufficient regarding desired energy savings.

However, different cooling system control strategies significantly affect the greenhouse's overall cooling demand. Table 1 shows that the higher cooling requirements reduce the energy saving potential of modern construction and control systems. If cooling demand is mainly caused by outer conditions (e.g. solar radiation, high outdoor temperatures – V3), these measures can reduce cooling demand about 60%. If cooling energy is mainly used to provide cold in the evening when the cubicles have to be cooled down (c.f. section 3), saving potential is significantly lower.

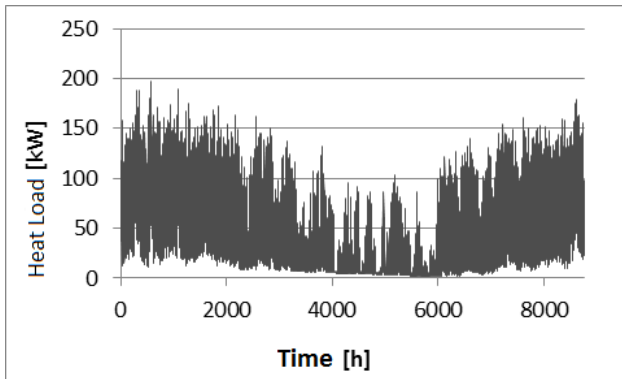


Figure 7: Heating load of alternative building and cooling strategy V1

Figure 7 and Figure 8 show the heating and cooling load curves with 15 minutes resolution which were calculated in the building simulation and are used as input data sets in following HVAC system simulation.

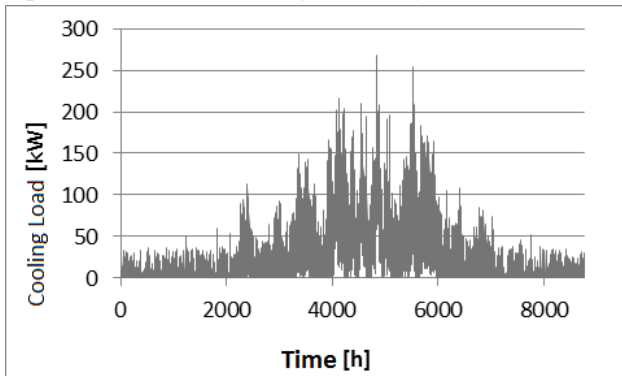


Figure 8: Cooling load of alternative building and cooling strategy V1

They are additionally used to proof HVAC system concept regarding individual components' requirements. Both characteristics represent worst-case scenarios for HVAC system operation, i.e. maximum heating and cooling load. Because the heat supply is covered by district heating, the required 200 kW heating power can easily be provided using a correctly dimensioned heat exchanger.

However, the maximum cooling power of HVAC system is mainly based on overall system characteristics. This way, the sum of the installed cooling power at suitable operating points of each component has to fulfill the total cooling power requirements in an adequate way. The overall installed cooling power in a worst-case-operating point (high recooling temperatures, low heating temperatures for absorption chiller) is about 215 kW. Figure 8 indeed shows a maximum cooling power demand of about 270 kW. However, this maximum cooling power is only required two or three times in a year and only for a few hours. Together with planned building internal cooling storage of 10 m³, the existing HVAC system concept should be able to fit all cooling requirements. Further system extensions are not necessary.

The final results of the simulations show that the alternative heating system concept can reduce overall ecological footprint about 68% in comparison to the reference system. This way 580 MWh oil-based heating will be replaced by 363 MWh from the district heating. Additionally, the specific carbon dioxide emissions are only half compared to the oil heating because the district heating system is heated by cogeneration power plants.

Evaluation of cooling system simulation requires more extensive analysis. The existing cooling system concept is divided into two parts, a high temperature (heat) part and a low temperature (cooling) part. Both parts are mainly connected via an absorption chiller.

The heat supply for absorption chiller is based on two different types of solar thermal collectors with an overall gross collector area of 194.7 m². Both collectors provide about 65 MWh heat over the whole year (55% direct flow and 45% heat pipe collectors). The difference between the collectors results from different efficiency characteristics and a comparatively high minimum flow temperature of 75°C.

The resultant solar collector utilization rate of 335 kWh/m² is quite good as normal rates are between 200 and 600 kWh/m². On the one hand, the required high flow temperatures significantly reduce collector efficiency. On the other hand the cooling system's heat demand perfectly matches solar heat availability.

Both solar collectors provide heat to the connected 10 m³ heat storage. Figure 9 shows resultant heat storage temperatures over the whole year. Because of the comparatively low heat demand of the absorption chiller during the winter time heat storage temperatures partly exceed maximum temperatures of 95°C. During these sunny winter days, solar collectors are shut down. This significantly reduces overall solar collector efficiency. Opposite to that solar heat is completely used by absorption chiller during summer time.

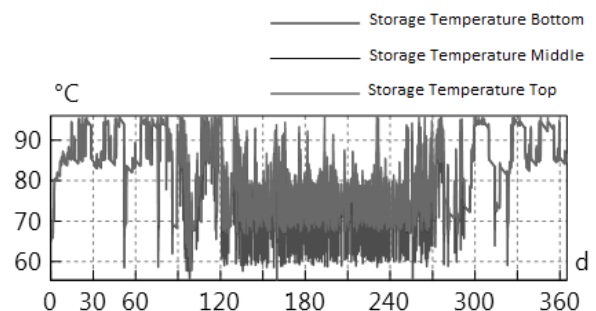


Figure 9: Simulated heat storage temperatures

Furthermore, Figure 9 shows a comparatively low temperature spread between bottom and top level of the storage tank. This behavior is caused by the storage tank mounted horizontally in the social building basement due to limited ceiling height of 3m. The low temperature spread further reduces solar collector and

absorption chiller efficiency because of the high return temperatures and high required volume flow rates. An alternative storage configuration with several cascaded storage tanks will help to solve this problem.

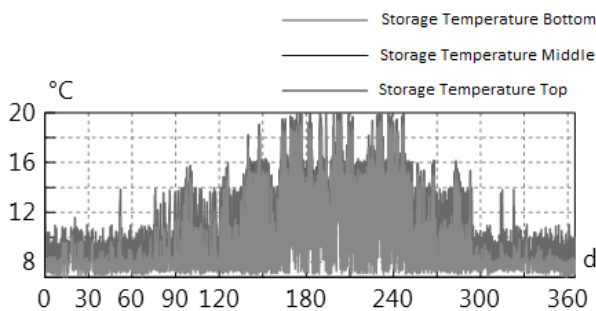


Figure 10: Simulated cooling storage temperature

The evaluated cooling system can provide enough cooling energy to keep cooling storage temperature at an adequate level (below 20°C) over the whole year. In cold winter times free cooling provides almost all cooling energy.

The basic cooling energy demand is supplied by the absorption chiller over the whole year. In times of higher cooling demand both peak power chilled water units are successively switched-on. Further cooling energy peaks are compensated by a 10 m³ sized cooling storage.

Again, the temperature spread in cooling storage is comparatively low. This is for the same reason as for the heat storage, horizontal mounting. Again, a cascade solution can significantly improve that situation.

Finally, the simulation results were used to validate desired saving potentials of the alternative system concept versus the reference.

- Heat:
 - Solar collectors: 65 MWh/a
 - District heating: 145 MWh/a
 - Absorption chiller: 208 MWh/a
- Cold:
 - Free Cooling: 19 MWh/a
 - Absorption chiller: 159 MWh/a
 - CWU 1: 41 MWh/a
 - CWU 2: 5 MWh/a
- Electrical energy:
 - Absorption chiller: 0.8 MWh/a
 - Hybrid cooler: 6 MWh/a
 - CWU 1 + dry cooler: 13 MWh/a
 - CWU 2 + dry cooler: 2 MWh/a

A priority control algorithm ensures that solar thermal collectors are mainly used to provide heat to the absorption chiller. However, the major heat demand is covered by the district heating because of

the high regeneration temperature requirements of the chiller.

The alternative HVAC system concept mainly replaces high electrical energy consumption of the reference system (chilled water units) with a higher district heating consumption supported by solar energy. However, simulation results show an overall reduction of ecological footprint of about 51% for the alternative concept versus the reference. This way, the alternative system concept fulfills the desired energy saving potential.

7 System Optimization

The basic task of system simulation was to validate the building planner's concept of 50% reduction of ecological footprint. However, extensive analysis of the model and the results made further optimizations possible.

One of these system optimization approaches directly affects overall solar thermal collector efficiency. Figure 9 shows that heat storage temperature exceeds the maximum level of 95°C mainly in winter time. This reduces solar collector efficiency because the system has to be shut down. However, more heat is still needed inside the building during that time periods, e.g. to heat social building. The main idea of optimization is to use solar thermal collectors for both, heating and cooling.

To test the potential, the required heating system components were added to the cooling system. Simulation results show that solar collector gains increase about 27% in the described system configuration. That way, the overall collector utilization rate can be increased from 335 kWh/m² to 432 kWh/m², a tremendous improvement of system efficiency.

However, extended use of solar energy also for heating reduces cooling system performance because the additionally required heat is provided by district heating. Ecological footprint of cooling system is reduced to 48.5% in comparison to the reference concept. The overall energy saving potential thus only increases from 62.7% to 64% (70% for heating).

8 Conclusion

This paper describes an innovative approach to integrate building and HVAC system simulation with Modelica in ordinary building planning process.

As assistance to existing design processes, the simulation results help to validate the planners' ideas. Furthermore, modeling and simulation knowledge can be used to provide further system optimization.

Designing complex HVAC systems with an increasing share of renewables and storage systems is no viable without any kind of system simulation. With its interdisciplinary background and easy-to-understand modeling approach, Modelica can help to

improve future building planning process. The efficiency of this simulation approach makes it more and more suitable even for smaller sized projects.

For building simulation Modelica's most important benefits are the wide range of suitable model libraries, its capability in combining different physical domains in one mathematical model description, and nevertheless the easy-to-understand modelling paradigms. This way, Modelica can help to close the gap between no more suitable static building calculation and numerical system simulation in building systems engineering.

Acknowledgements

The presented modeling and simulation results were developed in closed co-operation with GEFOMA GmbH Großbeeren and Saxonian Real Estate and Construction Management, Public Company.

References

- T. Schwan. Monitoring Concept and Validation of HVAC System Concept of Heat and Cold Supply for an Innovative Research Greenhouse in Leipzig. Project report, 2015.
- GEFOMA Großbeeren GmbH. IDIV greenhouse construction in Leipzig. Planning documents, 2015.
- VDI – Verein Deutscher Ingenieure. Cooling Load Calculation of Air-conditioned Rooms. VDI 2078 standard, 2007.
- A. Bertram, D. Wilms, A. Böttig, R. Rehrmann. Increase of energy-efficiency of energy screens. Final report, University of Applied Sciences Osnabrück, 2004.
- T. Schwan, R. Unger, B. Bäker, B. Mikoleit, C. Kehrer. Optimization of local renewable energy systems using automotive simulation approaches. 12th Conference of International Building Performance Simulation Association, Sydney, 2011.
- T. Schwan, R. Unger, B. Bäker, B. Mikoleit, C. Kehrer: Optimization-Tool for local renewable energy usage in the connected system: Building-eMobility; 8th International Modelica Conference, Dresden, 2011.
- T. Schwan, R. Unger, B. Bäker, B. Mikoleit, C. Kehrer, T. Rodemann: "Green Building" - Modelling renewable building energy systems and electric mobility concepts using Modelica, 9th International Modelica Conference, Munich, 2012.
- T. Schwan, R. Unger, C. Lerche, C. Kehrer: Model-Based Design of Integrative Energy Concepts for Building Quarters using Modelica. 10th International Modelica Conference, Lund, 2014.
- T. Schwan, R. Unger: AUTOmoble EnergieArchitektur – Final research project report. Dresden University of Technology, Dresden, 2013.
- M. Wetter: A Modelica-based model library for building energy and control systems, 11th International IBPSA Conference, Glasgow, 2009.
- Yazaki: Data sheet of absorption chiller machine WFC-SC 30. Data sheet, 2015.

Production Planning for Distributed District Heating Networks with JModelica.org

Håkan Runvik¹ Per-Ola Larsson¹ Stéphane Velut¹ Jonas Funquist² Markus Bohlin³ Andreas Nilsson³ Sara Modarrez Razavi³

¹Modelon AB, SE-223 70 Lund, Sweden, {hakan.runvik, per-ola.larsson, stephane.velut}@modelon.com

²Vattenfall R&D, 169 92 Stockholm, Sweden, jonas.funkquist@vattenfall.com

³SICS Swedish ICT, SE-164 29 Kista, Sweden, markus.bohlin@sics.se

Abstract

The short term production planning optimization problem for a district heating system is solved in two steps by integrating physics-based models into the standard approach. In the first step the unit commitment problem (UCP) is solved using mixed integer linear models and standard mixed-integer solvers. In the second step the economic dispatch problem is solved, utilizing the unit statuses from the UCP. This step involves dynamic optimization of non-linear physics-based models. Both optimizations aim at maximizing the production profit.

The modeling has focused on distributed consumption and production. Optimization results show that modeling of the district heating net impacts the production planning in several ways, with results such as reduction of production peaks and delay of costly unit start-ups.

The physics-based modeling and dynamic optimization techniques provide a flexible way to formulate the optimization problem and include constraints of physically important variables such as supply temperature, pressures and mass flows.

Keywords: district heating, physical modeling, distribution, optimization

1 Introduction

1.1 Background

The goal of production planning is to determine the most profitable scheduling of the different production units in a network, without violating operational constraints. It can be viewed as an optimization problem, which contains both continuous and discrete variables.

The operational statuses (on or off) of the different production units form the discrete decision variables of the optimization problem. The continuous decision variables are production unit loads and pump speeds.

The formulation also includes non-linear parts, such as turbine characteristics and steam properties. This

results in an optimization problem referred to as a mixed integer non-linear problem (MINLP). Currently, there are no known algorithms with predictable and robust performance for solving this kind of problem.

The predicted customer heat load during the optimization interval is the main input to the production planning problem. The prediction is often generated from weather forecasts and cannot be known exactly in advance. In this paper manually generated predictions are used, mostly assuming perfect predictions, but formulations where uncertainties are included are also investigated. The implemented optimization model is based on the units and network distribution of the Uppsala district heating network, with special emphasis on the modeling of the cogeneration plant KVV.

The standard method to circumvent the difficulties of solving a MINLP problem in a production planning formulation is to simplify the modeling considerably. By linearizing plant models and reducing the network model to only contain energy flows, a linear optimization formulation is obtained instead. This kind of problem is called a Mixed Integer Linear Problem (MILP) and can be solved using standard techniques. Previous work based on linear plant models include (Arroyo and Conejo, 2004), where a method to formulate start and stop trajectories is presented and (Rolfman, 2004), where a heat storage strategy based on the variations in electricity price is presented. In (Rong, *et al.*, 2008) an improved algorithm for the unit commitment problem is presented.

1.2 Proposed Approach

The separation of the optimization problem into the Unit Commitment Problem (UCP) and the Economic Dispatch Problem (EDP) part presents an alternative solution to the problem of creating a robust optimization formulation of the production planning problem. The two optimization problems are solved in series, a method previously implemented in (Velut *et*

al, 2013). The modeling and optimization efforts are conducted in the following manner:

- UCP: A linear optimization formulation is obtained by approximating the district heating network using piecewise linear models. The problem is solved using a MILP solver with the status signal for each production unit being the main result.
- EDP: A representation of the district heating network is created using physical modeling. Smoothed versions of the status signals from the UCP are implemented in the model, so that only continuous variables are present in the optimization formulation. By solving the optimization problem, the load for each unit is decided.

There are several benefits with including physical modeling in the optimization formulation. The optimization model becomes highly accurate when physical laws such as mass and energy balances are used to describe the units of the network. It also makes it possible to optimize physically relevant variables that effect the plant economics such as supply temperatures and mass flows. The possibility to impose constraints on these variables, based on the physical and operational limitations of the real system is another advantage.

In order to solve the EDP, the optimization problem is discretized into a Non-Linear Programming (NLP) problem using the so-called collocation method (Magnusson, 2012). Different solvers for NLP problems exist, in this work the open-source solver IPOPT (Interior Point Optimizer), see (Wächter and Biegler, 2006), was used. In previous projects the authors have used this method for dynamic optimization of a carbon capture plant (Åkesson *et al*, 2011) and, more notably, for short-term production planning of district heating (Velut *et al*, 2013).

2 Modeling

2.1 Uppsala District Heating Network

The production units and network distribution of the Uppsala district heating network were used as models when the production planning setup was created in this work. The main production unit in this network is the cogeneration plant KVV located at the production site Boländerna. The KVV has a production capacity of approximately 250 MW heat and 130 MW electricity. Other important units in the system include several oil boilers, a waste incineration plant, and an accumulator.

2.2 Discrete Optimization Model

The models used in the UCP are formulated in Python using the Pyomo modeling language. The models are linear and coarse and are mainly describing energy and energy flows.

2.2.1 Cogeneration Plant KVV

The KVV is modeled using a polytope in the space of electricity, heat and return temperature, which is displayed in Figure 1. This means that for each return temperature the polytope provides an area in the electricity-heat plane which the electricity and heat production is confined to. The KVV model in the EDP, which is summarized in section 2.3.1, was used to generate the polytope. The fuel consumption U_{KVV} for a certain electricity production P_{el} and heat production Q_{KVV} is calculated using the efficiency η_{KVV} according to

$$U_{KVV} = \frac{Q_{KVV} + P_{el}}{\eta_{KVV}} \quad (1)$$

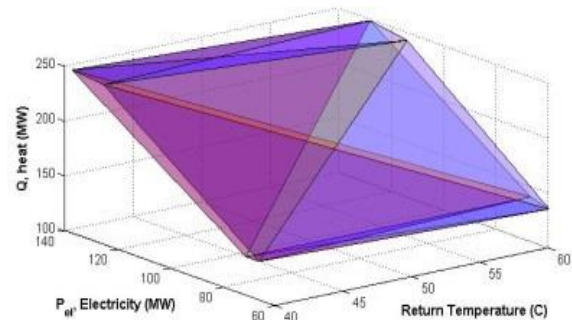


Figure 1. Polyhedron representing the operating regions of the cogeneration plant.

2.2.2 Other Production Units

For units that only produce heat, the relation between produced heat Q_{unit} and fuel consumption U_{unit} is given by

$$U_{unit} = \frac{Q_{unit}}{\eta_{unit}} \quad (2)$$

2.2.3 Accumulator

The accumulator works as an integrator, where the stored energy E_{acc} is determined by

$$E_{acc}[t] = E_{acc}[t - 1] - hQ_{acc}[t - 1] \quad (3)$$

where $Q_{acc}[t - 1]$ is the energy flow to or from the accumulator and h is the sampling period.

2.2.4 Pipe Model

In order to represent the influence of the transportation of the district heating water, a pipe model containing a fixed time delay and a heat loss model is used. The heat loss from a pipe section, \dot{Q} , is determined using the outdoor temperature and is based on the following formula, describing the heat transferred from an underground cylinder with temperature t_0 , when the ground temperature is t_s (Sundén, 2006).

$$\dot{Q} = \frac{2\pi\lambda L(t_0 - t_s)}{\ln\left(\frac{2N}{D} + \sqrt{4\left(\frac{N}{D}\right)^2 - 1}\right)} \quad (4)$$

The other parameters of this equation is summarized in Table 1.

Table 1. Heat transfer parameters.

Parameter	Interpretation
L	Pipe length
D	Pipe diameter
N	Pipe depth
λ	Soil heat transfer coefficient

2.3 Continuous Optimization Models

The EDP modeling was performed in Dymola, where Modelica models representing the different units and components of the district heating network, were created.

Two different water media models are implemented, an advanced model using polynomials to approximate IF97 reference functions, and a simple model where the specific heat capacity and density of the water are constant. The advanced medium model is used in the vapor cycle of the KVV, while the simple medium model is used to represent the district heating water.

2.3.1 Cogeneration Plant KVV

The goal of the modeling of the KVV is to capture how the produced heat and electricity depends on the plant load, return water temperature and mass flow. For this reason the modeling efforts have been directed towards the vapor cycle. The entire cycle is however not included in the model, instead boundary conditions

have been implemented using the following assumptions:

- The boiler outlet vapor characteristics (pressure and enthalpy) are constant and the mass flow is proportional to the plant load.
- The condensate leaving the condensers is at saturation pressure.
- Bleed streams from low pressure turbines are represented by a lumped pressure drop and a fixed pressure boundary.

A schematic illustration of the model of the KVV is displayed in Figure 2. A summary of the main components used in this model is presented below.

- Turbine: An isentropic efficiency parameter is used to calculate the outlet enthalpy and the mechanical work, while Stodola's law determines the relation between mass flow and pressure drop. The electrical output is calculated using mechanical and electrical efficiencies.
- Condenser: By considering the difference between incoming water temperature and the saturation temperature a heat flow rate to the district heating water is calculated. This heat flow rate determines the condensation rate and consequently the bleeding flow from the turbine stages.
- Control volume: Dynamic mass and energy balances are used to model a control volume. The equations are formulated using pressure and enthalpy as states, which requires partial derivatives of density with respect to enthalpy and pressure.
- Pressure loss: A quadratic loss function is used to relate the mass flow to the pressure drop.
- Reheater: An ideal representation of the reheating

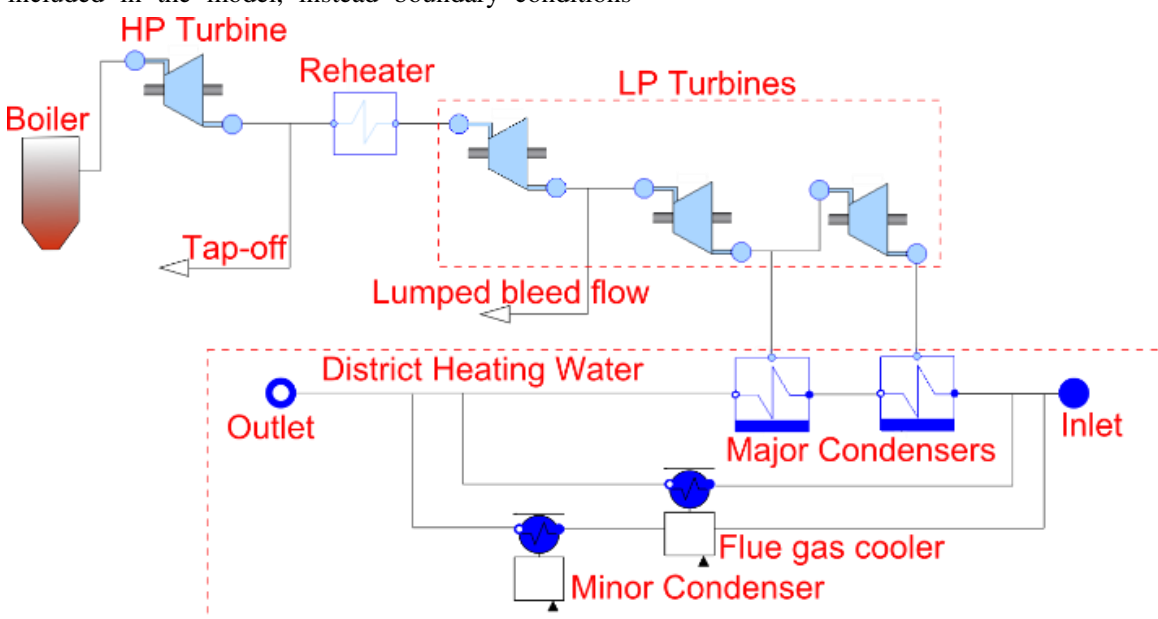


Figure 2. Schematic overview of the cogeneration plant model.

in the plant, as the outlet temperature is constant and determined by a parameter value.

2.3.2 District Heating Network Models

The models used to represent all units in the district heating network, except for the KVV, are summarized below.

- Heat production unit: The heat production from other units than the KVV is modeled empirically, adding heat to the district heating water proportionally to the firing power.
- Customer model: The mass flow through each customer is determined by the customer load model. The difference between the supply temperature and the predefined return temperature, which is based on the outdoor temperature, provides the mass flow based on the heat demand.
- Accumulator: A finite volume approximation is used where buoyance effects are neglected, i.e. no mixing is assumed when the accumulator is not charging or discharging. Heat losses are also neglected.

2.3.3 Pipe model

The production units and the customers are connected using pipe models. These are modeled using a combination of a standard finite volume implementation and a fixed delay of the temperature profile. The two components are connected in series, together with a heat loss component using the same heat dissipation equation as in the UCP pipe.

The goal of combining a fixed delay with a finite volume model is to capture the main characteristics of the pipe without having to use a model with very many pipe segments, something that would increase the complexity of the optimization problem considerably. It is a compromise between using only a fixed delay, which would result in incorrect delay times when the mass flow is varying, and using a fixed volume implementation with few volume segments, which would result in numerical dissipation. The ratio between the fixed delay and the finite volume pipe volume is decided based on the range of mass flows that will occur in each pipe and the accepted delay time error for the boundaries of this range.

2.4 Network Representation

The distribution of the customers and production units in the Uppsala district heating network is modeled using a one-dimensional approach. The network description is based on the setup presented in (Saarinen and Boman, 2012), where the customer distribution as a function of the delay time is determined. Compared to that model the setup in the optimization models is simplified further and only includes three customers. In Figure 3 a schematic representation of the implemented network structure is displayed.

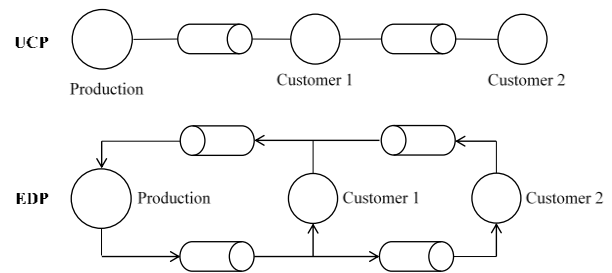


Figure 3. Schematic representation of network structure used in the discrete (upper structure) and the continuous optimization (lower structure).

3 Optimization Tools

3.1 Discrete Optimization

The UCP problem was formulated in Python using the Pyomo modeling language. Two different solvers were used for solving the UCP, the commercial solver Gurobi (Gurobi Optimization, 2015) and the open source package GLPK (Makhorin 2012).

3.2 Continuous Optimization

The optimization problem for the EDP was formulated using the Optimica language, extending the Modelica models describing the system. The open-source JModelica.org platform (Modelon AB, 2014) was used to translate the formulation into an NLP and this problem was solved using the Interior Point Optimizer (IPOPT), see (Wächter and Biegler, 2006). FMUs were used for initial trajectory simulations.

4 Optimization Formulation

4.1 Cost Function

In both the EDP and the UCP the goal is to maximize the economical profit. Incomes from selling heat and electricity, fuel costs and maintenance costs are therefore the main parts of the cost functions in the two optimization formulations. Only constant heat, electricity and fuel prices are considered and additional costs, such as pump costs are not considered in the model. In the UCP costs for starting and shutting down production units are also included in the cost function.

Additional terms must be added to the cost functions for numerical reasons. In the UCP one can easily obtain multiple solutions. To avoid this problem a small cost penalizing production unit load changes have been added. In the EDP a minor cost on input derivatives must be implemented for regularity reasons.

4.2 Degrees of Freedom

In the UCP the heat production and the status of each unit are decision variables, as well as the KVV electricity production and the energy flow to or from the accumulator.

The decision variables for the EDP are similar to those in the UCP, but with a few key differences. Firstly, the status of each unit is fixed in the EDP. Secondly, it is not the heat production of each unit that is the decision variable, but the load change. This is achieved by introducing equations of the form

$$U_{unit}(t) = \int_t \dot{U}_{unit}(t) dt \quad (5)$$

The same formulation is used for the accumulator, but here another difference is also preset, as it is not the energy flow, but rather the district heating water mass flow that is controlled.

4.3 Constraints

Constraints represent an important part of the optimization formulation. In this section the most important constraints in the UCP and EDP formulations are presented.

All production units in the UCP and the EDP have constraints on their productions and their production change rates, corresponding to the limitations of the real plants. For the accumulator there are similar constraints defining the minimal and maximal amount of energy that can be stored, and how fast the energy level can change. To prevent emptying of the accumulator at the end of the optimization interval, an additional constraint of the form

$$E_{acc}[t_f] \geq E_{acc}[t_0] \quad (6)$$

is used in the UCP. Here t_0 and t_f represents the endpoints of the optimization interval. In the EDP an accumulator constraint based on the UCP accumulator energy at the end of each optimization interval is used.

When a production unit changes status the heat production must follow specific start and stop trajectories, denoted $Q_{unit,start}[t]$ and $Q_{unit,stop}[t]$, respectively. In the UCP this is implemented using constraints of the form

$$Q_{unit}[t] = Q_{unit,start}[t], \quad t \in [t_{start}, t_{start} + t_{startdelay}] \quad (7)$$

$$Q_{unit}[t] = Q_{unit,stop}[t], \quad t \in [t_{stop}, t_{stop} + t_{stopdelay}] \quad (8)$$

where $t_{startdelay}$ and $t_{stopdelay}$ are the durations of the constraints.

In the EDP the trajectories must not be followed exactly. Instead upper and lower constraints are used to confine the production to be close to the trajectory are used.

In the EDP more constraints are present, limiting e.g. mass flows, temperatures and pressures in different

components. For a more detailed description, see (Larsson *et al*, 2014).

5 Optimization Example

Several test cases of varying complexity were developed to evaluate the production planning strategy. In this paper the main results from the most realistic case are presented.

5.1 Optimization Settings

A sampling interval of 30 minutes for UCP optimization and 20 minutes for EDP optimization is used in all optimization cases. The optimization interval is between one and four days in the UCP and between 20 and 24 hours in the EDP. The difference in optimization horizon is a result of the different objectives of the optimization problems; the UCP results determine long term plans while the EDP handles faster dynamics.

The customer load profile consist of a base load with two load peaks per day, representing the typical heat demand of a residential area.

5.2 Test Case

In this test case, the heat load profile is increasing linearly, with load peaks superimposed. The load profile during the first day of the scenario is displayed in Figure 4. The problem setup involves three production units, the accumulator, and customers. One of the production units, the waste incineration plant AFA is running with maximal load throughout the optimization. The KVV is also running at all times, but the load is a decision variable. The final producer is the Husbyborg oil boiler. This unit is initially turned off, but must eventually be started as the customer heat demand is increasing.

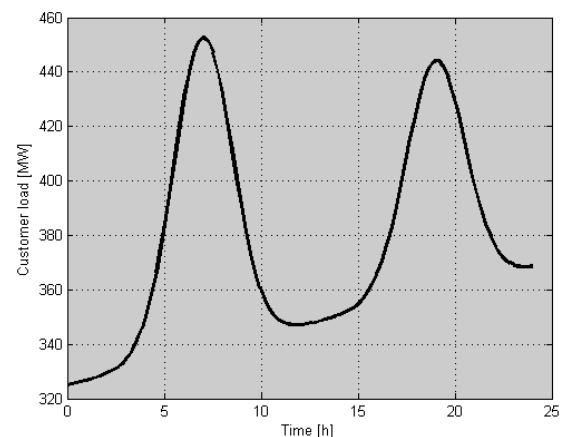


Figure 4. Customer load profile during day one.

Two subcases are considered, in the first one a point-wise network representation with one customer is implemented and in the second one a distributed network is used. The optimization interval is four days

for the UCP. For the EDP, this period is divided into five separate optimizations for the first subcase, and six optimizations in the second subcase.

5.2.1 Optimization Results

The results from the test case are displayed in Figure 5 to Figure 8. The most important result is the difference in start-up time for the oil boiler, depending on which network topology that is considered. By including the distribution of the customers in the optimization formulation it is possible to delay the start-up with nine hours, from hour 30.5 to hour 39.5. The difference is explained by the reduced production peaks caused by the difference in time delay between the different customers.

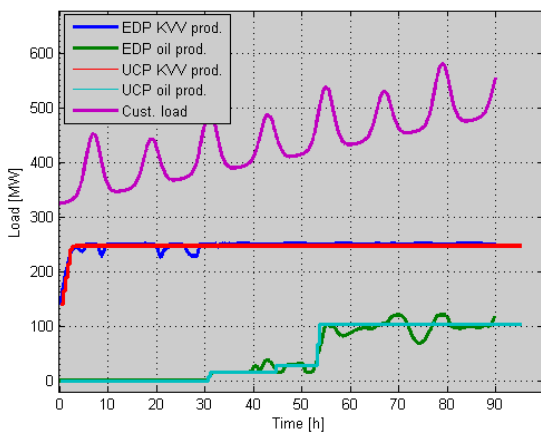


Figure 5. Customer load and heat production for different units using a point-wise network. Results from the discrete (UCP) and the continuous (EDP) optimizations are compared.

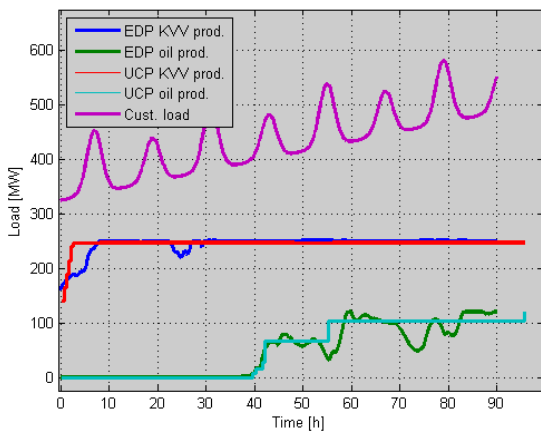


Figure 6. Customer load and heat production using a distributed network. Results from the discrete (UCP) and the continuous (EDP) optimizations are compared.

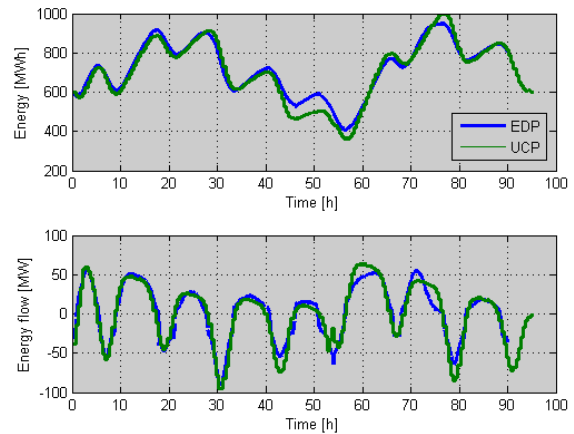


Figure 7. Accumulator usage in the point-wise network case. Results from the discrete (UCP) and the continuous (EDP) optimizations are compared.

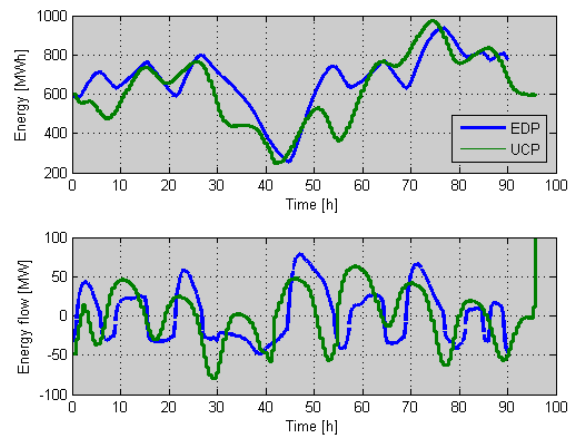


Figure 8. Accumulator usage in the distributed network case. Results from the discrete (UCP) and the continuous (EDP) optimizations are compared.

One can also see that there are some differences in the behavior of the EDP results compared to the UCP. Especially the signal describing the heat production of the oil boiler contains oscillations in the EDP results, which are not present in the UCP results. The oscillations are a result of the more detailed optimization model used in the EDP, which includes faster dynamics, and the shorter optimization horizon used for the EDP. The more detailed modeling makes it possible to utilize effects such as heat storage in the pipes and mass flow dependent delay times. This results in an optimal strategy that contains faster load variations.

The shorter optimization horizon introduces some transient behavior at the end of each optimization interval for the EDP, as the optimization attempts to use the free heat stored in the network. To counteract this, the final part of each optimization was disregarded, but nonetheless some transient behavior based on this effect can be observed. By implementing

the EDP as an MPC, disregarding more of the optimization results, this unwanted effect would be removed. However, the EDP results are in general of higher quality than the UCP results due to the more detailed modeling. This means that they are more physically relevant, and also more optimal for the actual structure of the district heating network.

Another notable feature of the optimization results is that the accumulator is used to compensate for the load variations, while the production plants are mostly running at constant load.

The continuous formulation of the problem above contains 307 variables and 33 states, while the transcribed NLP formulation contains approximately 70 000 variables. Using a standard laptop with 8 GB RAM and four 2.6 GHz CPUs, the optimization problem was solved in less than ten minutes.

5.3 Conclusions from Other Test Cases

- The district heating water mass flow is typically maximized and the supply temperature is correspondingly minimized. This results in a maximization of the KVV electricity production.
- Pump limitations, customer supply temperature and condenser pressure can all be limiting the district heating highlighting the benefits of thorough physical modeling of the system.
- By introducing a pipe model to represent the customer distribution the mass flow dependency of the delay time can be captured.
- The importance of the delay time between customer and producer is highly depending on whether temperature or mass flow changes are used to compensate for heat load variations. When only the mass flow changes the delay time is irrelevant as water is incompressible.
- The possibility to use the network as an accumulator follows from physical modeling of the distribution network.

6 Conclusions

In this paper an extension of the approach for short-term production planning presented in (Velut *et al*, 2013) has been proposed. The economic dispatch problem is solved with JModelica.org, using non-linear optimization of physical models. The method have been investigated using data from the district heating network in Uppsala, Sweden.

The derived optimization strategy involves minimization of the district heating water supply temperature and, correspondingly, maximization of the mass flow. By considering constraints on variables such as pump speed, condenser pressure and customer temperature the limitations of the real system have been included in the formulation and the effect of these constraints can be seen in the optimization results.

The network distribution is included in the optimization model using physical pipe models and a simplified topology. By using this network model the different time delays for different customer groups is included in the model. In computation experiments, the distributed customer increased the economic profit by lowering production peaks and utilizing heat accumulation in the network.

Acknowledgements

Grateful acknowledgments to Värmeforsk, “The Swedish Thermal Engineering Research Institute” and Energimyndigheten, “The Swedish Energy Agency”, for providing financial support (project 38155).

References

- J. Arroyo and A. Conejo. Modeling of start-up and shut-down power trajectories of thermal units. *Power Systems, IEEE Transactions on*, 19(3): 1562–1568, August 2004. doi: 10.1109/TPWRS.2004.831654
- Gurobi Optimization. 2015. Accessed 29 April 2015 <<http://www.gurobi.com>>.
- P.-O. Larsson, S. Velut, H. Runvik, S. Modarres Razavi, A. Nilsson, M. Bohlin och J. Funkquist. Decision Support for Short-Term Production Planning of District Heating using Non-linear Programming. Värmeforsk, 2014.
- Fredrik Magnusson. Collocation Methods in JModelica.org. Master’s thesis. 2012
- Andrew Makhorin. 2012. GNU Project. Accessed 22 October 2014. <<https://www.gnu.org/software/glpk/>>.
- Modelon AB, 2014. Accessed 22 October 2014. <<http://www.jmodelica.org>>.
- B. Rolfsman. Combined heat-and-power plants and district heating in a deregulated electricity market. *Applied Energy*, 78(1):37 – 52, 2004. doi:10.1016/S0306-2619(03)00098-9
- A. Rong, H. Hakonen, and R. Lahdelma. A variant of dynamic programming algorithm for unit commitment of combined heat and power systems. *European Journal of Operational Research*, 190(3):741–755, November 2008. doi:10.1016/j.ejor.2007.06.035
- L. Saarinen, and K. Boman. Optimized district heating supply temperature for large networks. Värmeforsk, 2012.
- Bengt Sundén. Värmeöverföring. Studentlitteratur, 2006
- S. Velut, P.-O. Larsson, J. Windahl, K. Boman, and L. Saarinen. Non-linear and Dynamic Optimization for Short-term Production Planning. Värmeforsk, 2013.
- 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. doi: 10.1007/s10107-004-0559-y
- J. Åkesson, C. Laird, K. Lavedan, K. Pröls, H. Tummescheit, S. Velut and Y. Zhu. Nonlinear Model Predictive Control of a CO₂ Post-Combustion Unit. *Chemical Engineering Technology*, vol. 35, no 3, pp. 445-454, 2011. doi: 10.1002/ceat.201100480

Hardware-in-the-Loop-Simulation of a Building Energy and Control System to Investigate Circulating Pump Control Using Modelica

Georg Ferdinand Schneider¹ Jens Oppermann² Ana Constantin³ Rita Streblov³ Dirk Müller³

¹Fraunhofer Institute for Building Physics, Systems Integration Group, Fürther Straße 250, 90429 Nürnberg, Germany, georg.schneider@ibp.fraunhofer.de

²WILO SE, Group Research and Technology, Nortkirchenstraße 100, 44263 Dortmund, Germany, Jens.Oppermann@wilo.com

³RWTH Aachen University, Institute for Energy Efficient Buildings and Indoor Climate, Mattheustraße 10, 52074 Aachen, Germany, {aconstantin,rstreblov,dmueller}@eonerc.rwth-aachen.de

Abstract

This paper presents an application of the hardware-in-the-loop-method to a building energy and control system. We focus on investigating the interaction of a real circulating pump with the hydronic network of a virtual building energy and control system. For real-time simulation the building envelope is modelled using the Modelica-based library AixLib. With the presented setup model-based designed control algorithms are tested directly on real hardware. The performance of the presented and implemented setup is evaluated by comparing simulated results with experimental hardware-in-the-loop-simulation data. The main focus of this work is to evaluate the application of the method towards bridging the gap between model-based design and commissioning of energy efficient control for heating ventilation and air conditioning (HVAC) components.

Keywords: Hardware-in-the-loop (HIL); Building Energy and Control System; Model-Based Design

1 Introduction

There is an ongoing research effort towards developing tools and processes to improve the design of building energy and control systems (BECSs) in order to reduce the environmental impact of existing and future buildings while improving their indoor comfort (Wetter, 2011a). With the ongoing reduction of the energy demand for heating and cooling of buildings, e.g. better insulation, research is focussing on increasing the energy efficiency of auxiliary HVAC equipment such as circulating pumps. Additional pressure is put on equipment manufacturers by the European Union by introducing mandatory eco-design requirements for energy-using products (EU, 2009).

To achieve these efficiency goals is an inherently difficult task as BECS typically are comprised out of heterogeneous components which have to be regarded simultaneously on different temporal and spatial scales (Wetter, 2011a). An approach to design control algorithms for components of these systems in an integrated way is the application of model-based design methods. Equation-based object-oriented modelling languages, e.g. Modelica, allow modelling these systems consistently (Wetter, 2011a). In Modelica ready-to-use libraries exist to perform whole building energy simulation for model-based design (e.g. Constantin et al., 2014; De Coninck et al., 2014; Wetter et al., 2014).

However, difficulties occurring during the transition process from design to commissioning stage of model-based control algorithms remain unsolved. The transition process can be time and cost consuming and a feedback of newly generated information to the design process is seldom possible.

Increasing circulating pump efficiency is mainly driven by implementing control algorithms into the application controller of pumps, which consider the interaction between the pump and its surrounding BECS. For the case of control algorithms for circulating pumps we observed the following obstacles:

- An error-prone reimplementation of the designed control algorithms is necessary at each iteration, as operating systems and programming languages differ between simulation environments and real hardware application controllers;
- Testing of control algorithms for circulating pumps requires detailed measurements of pump prototypes in test-buildings over time periods of days and weeks.

Hardware-in-the-loop-(HIL-) simulation is a method to bridge the gap between simulation and real hardware

by coupling simulated and real parts of a system. The HIL-method is well-known in science and technology and several applications based on Modelica are reported, e.g. in automotive, food and microcontroller industries (Winkler and Gühmann, 2006; Gäfvert et al., 2008; Bonvini et al., 2009). In the buildings industry the HIL-method is frequently used for testing and commissioning of building control systems (e.g. Xu et al., 2004). Chen et al. (2012) present a multi-domain test setup to investigate building control and HVAC-systems with simulated boundary conditions. Applying the HIL-method to a circulating pump in the buildings domain is, to the best of our knowledge, a novelty.

The contribution aspect of this paper is the presentation of a successful application of the HIL-method to a BECS to investigate control algorithms for HVAC components. By this for the application of a circulating pump,

- model-based designed algorithms may be tested directly on real hardware without reimplementation and
- different building types can be examined on a single hardware setup by using different models.

After the introduction we present a brief description of the HIL-method in section 2. A HIL-concept for a BECS is presented in section 3. Finally, in section 4 we evaluate the performance of an implementation of the HIL-concept by a comparison of measured data and simulated results.

2 Methodology

This section gives a brief overview of the HIL-method. In HIL-simulation real hardware is coupled to virtual components of a system which are simulated in real-time (Maclay, 1997). The loop comprises of providing the measured behaviour of real components as boundary conditions to virtual models of a system and emulating the simulated reaction at every time step to the hardware components.

The HIL-method offers several advantages in its application (Maclay, 1997). Despite the decreased development times in today's development cycles it is necessary to keep the established high standards in safety and quality of products and processes. HIL supports and simplifies research and development processes of complex systems by giving the possibility to test components of heterogeneous systems without danger for humans, environment or plants by malfunctioning or unexpected behaviour (Maclay, 1997). HIL-simulation often is easily applicable as models available from model-based development processes may be reused for HIL-simulation with minor changes. Nevertheless the accuracy of the used models has to be carefully assessed for a successful application of the method. When developing an application of the method it is necessary

to evaluate the dynamic performance of the designed emulation facilities to ensure a proper feedback of the simulated behaviour to the real components.

A HIL-system can be divided into three parts: simulation, emulation and hardware level (Chen et al., 2012). At the simulation level models of the virtual components are provided and simulated in real-time. The emulation level actually enables the coupling of real and virtual system components. It includes devices and facilities to measure the hardware behaviour and to emulate the simulated behaviour of the virtual components. A real-time capable data interface is necessary to exchange data between the data acquisition devices which measure the physical quantities and the simulation environment at run time. At the hardware level real components of a complex system are installed in an emulator.

3 Application of HIL-Method

In this section we demonstrate the application of the HIL-method to a BECS to investigate control algorithms for circulating pumps. The details and a real setup of a HIL-concept for a circulating pump are presented. With the proposed HIL-setup it is possible to examine the interaction of a real circulating pump in a hydraulic network of a virtual BECS. In our specific case both the model of the BECS and the algorithms for circulating pump control are implemented in one simulation environment. The coupling of the simulation environment and the hardware allows to directly execute the control algorithms on a real pump. A scheme of the realized HIL-concept is presented in Figure 1.

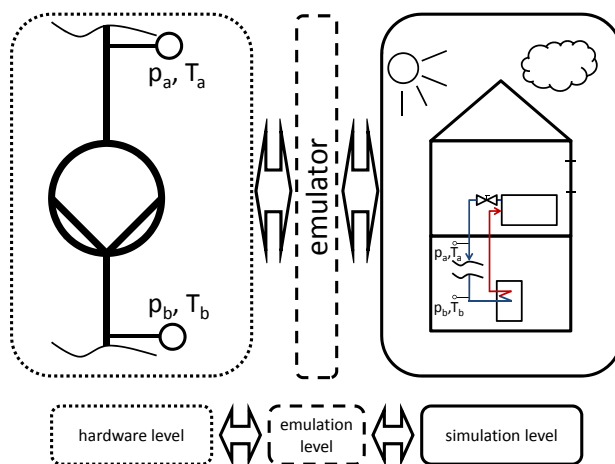


Figure 1. Scheme of the developed HIL-concept of a real pump in a virtual BECS. The boundary conditions pressure $p_{a,b}$ and temperature $T_{a,b}$ up- and downstream of the pump are exchanged via a data interface (arrows).

We divide the HIL-system in a hardware, emulation and simulation level. At the hardware level a real pump is

installed in a hydraulic circuit at a testbench. The boundary conditions of the pump are characterized by pressure $p_{a,b}$ and the temperature $T_{a,b}$ up- and downstream of the pump. The downstream boundary conditions are measured and given as input to a BECS model at the simulation level. The upstream values are calculated by a real-time simulation of the BECS and are emulated at the emulation level to the real pump.

The following sections describe the simulation, emulation and hardware level of the proposed HIL-concept in detail.

3.1 Simulation Level

We choose the Modelica-based library AixLib to model one and multiple family dwellings for simulation in Dymola (Dymola, 2013). The library is open source and accessible from an online repository (<https://github.com/RWTH-EBC/AixLib>). We utilize ready-to-use models of the library to describe the building envelope. To model the HVAC and control system we use in this application models from the library described in Müller and Hosseini Badakhshani (2010). The full details of the used models is not in the scope of this paper. For a detailed description please refer to the mentioned references. The main modelled aspects are:

- Each room model has one air node modelled as a single, perfectly mixed air volume;
- The air volume exchanges heat to its enclosures by convective, conductive and radiative heat transfer;
- A wall model consists of layers with different physical properties each; transient heat conduction through each layer is approximated using a one dimensional approach;
- The model of the heating system consists of radiators with thermostatic valves for each room, pipes, a boiler with night setback and an expansion tank;
- A weather file providing weather data of a typical reference year (TRY) for locations within Germany (DWD, 2013);
- User occupancy, ventilation and infiltration are considered.

Figure 2 illustrates schematically the structure and components of the modelled BECS of a one family dwelling. The models of AixLib are currently under evaluation by using a set of standardized tests provided by the American Society of Heating, Refrigerating and Air-Conditioning Engineers to validate building simulation models (ASHRAE, 2004). Additionally, we statically validated the building model by comparing the simulated results with results of the design heat load calculated according to EN 12831 (CEN, 2003) using the commercial tool SOLAR-COMPUTER (Solar-Computer, 2012).

For the HIL-system we adapt models developed for

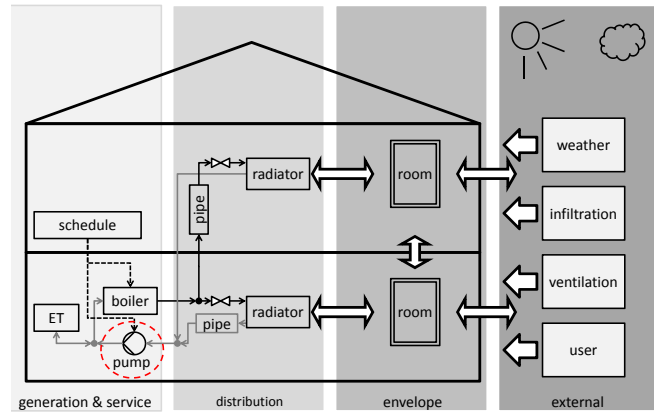


Figure 2. Scheme of the building model. Pump model (dashed, red) is substituted by two boundary conditions with inputs for the case of HIL-simulation (ET – expansion tank).

model-based design of control algorithms for circulating pumps by modifying them for HIL purposes. To clarify the model structure the top level of the simulation model is presented in a scheme in Figure 3. It comprises of a building envelope and HVAC system model and the weather input. Instead of a pump model two hydraulic boundary conditions (fluid source, fluid sink) are implemented. Together with the building models a TCP/IP-based data interface is implemented and simulated as well as the pump control algorithms. Via the data interface the value for the current pressure difference Δp , the current volume flow rate of the hydraulic circuit Q , the current set point of the pump y_{pump} and the set point for the motor-driven valve y_{valve} are exchanged at runtime. The number of equations of the resulting model is typically in the range of ten thousand.

To simulate the described models in real-time we use Dymola with default settings and its capability to synchronize with an external Dynamic Data Exchange (DDE) server for (soft) real-time simulation.

3.2 Emulation Level

The emulation system acts as an interface between the real hardware and the virtual components. It performs two tasks: first, it offers facilities to measure physical quantities and transmit the values to the simulation environment; second, it has the capability to convert the simulated values back into real physical quantities (Chen et al., 2012).

In this section we present a solution to emulate the hydraulic boundary conditions and a data interface which couples the LabVIEW-based testbench software (NI, 2015) and Dymola. A scheme of the system is displayed in Figure 4. We assume negligible temperature drop across the pump and therefore the facilities for the temperature emulation are not discussed here.

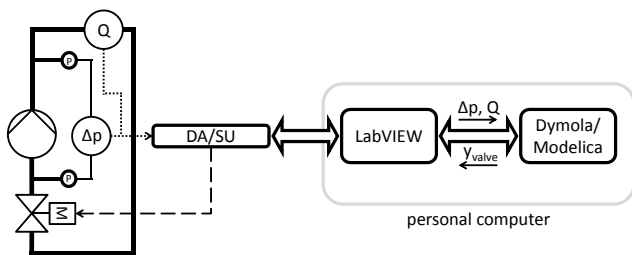


Figure 4. Scheme of the testbench setup (DA/SU – data acquisition/ switch unit). Data exchange between LabVIEW and Dymola/Modelica via a TCP/IP-based interface, Δp – pressure difference, Q – volume flow rate in hydraulic circuit, y_{valve} – set point motor-driven valve.

Emulation of Hydraulic Boundary Conditions:

To establish a loop the pressure difference Δp across the pump is measured for one time step by a differential pressure sensor with precision of 1 % of the measured value and is given to the simulation as input. Then the model is simulated for this time step and the corresponding total volume flow rate Q_{set} of the building is calculated. This volume flow rate needs to be converted into a physical quantity by the testbench.

To perform this task we measure the volume flow rate Q in the hydraulic circuit of the testbench by a magnetic flow meter with a precision of 0.5 % of the measured value and a response time of 0.125 s. We implement a feedback control with the simulated total volume flow rate as set point. A PI-controller with limited output implemented from the Modelica Standard Library is used to control the volume flow rate Q through an adjustable, motor-driven valve with a nominal stroke of 5.5 mm, a resolution of 1:100 and a positioning speed of 13.6 mm/s.

The dynamic performance of the described volume flow rate control system can be assessed from results of an exemplary step response presented in Figure 5. Measurements are performed at a sample rate of 1 s. From the results we measure a static deviation of ± 3.5 l/h and a settling time of 200 s.

We assess the dynamics and the precision of the system

to be sufficient for the considered application. Time constants in thermal systems of buildings are in the range of 1000 seconds which is one magnitude larger than the determined settling time.

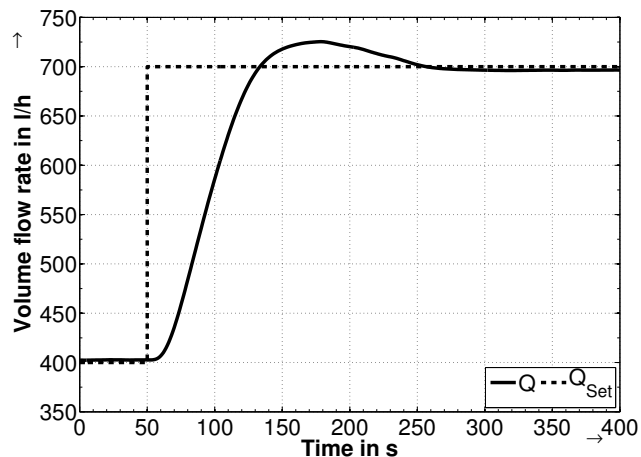


Figure 5. Step response of volume flow rate control system, Q_{set} – Set point of PI-control, Q – measured volume flow rate in hydraulic system, sampling rate: 1 s.

Socket-Based Data Interface:

A prerequisite of the development of the HIL-testbench is the usage of a proprietary, LabVIEW-based data interface to enable access to the low-level pump control with the possibility to execute model-based designed control algorithms without any reimplementation. A disadvantage of this platform at the point of implementation is that no ready-to-use data interface to exchange data at run time to the simulation environment was available.

We followed a straightforward approach and implemented a real-time capable socket-based data interface to enable data exchange with the simulation environment at run time. The data interface is based on the Winsock API (Microsoft, 2015) which allows to exchange character strings via network sockets. The interface is implemented in C and uses a set of functions to exchange character strings using the TCP/IP protocol. To include the additional C code we use the `external C` construct of

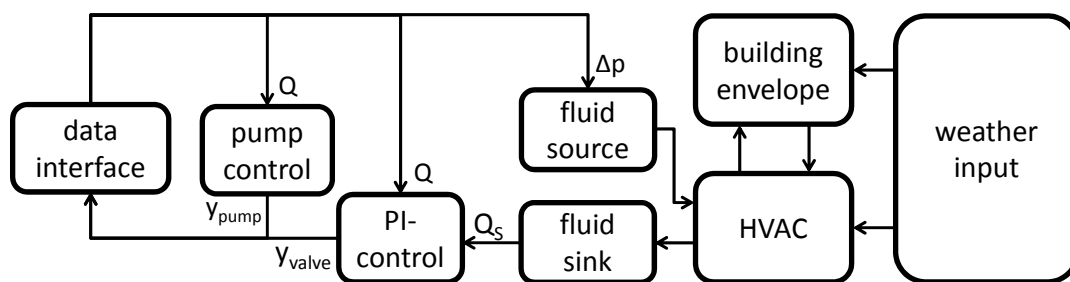


Figure 3. Scheme of the top level of a Modelica model for HIL-simulation, Δp – pressure difference, Q – volume flow rate in hydraulic circuit, Q_s – simulated volume flow rate, y_{pump} – set point pump, y_{valve} – set point motor-driven valve.

Modelica and compile the code with the Modelica code of the model.

Solutions for data interfaces for co-simulation and HIL already exist (e.g. Bellmann, 2009; Wetter, 2011b). We discovered in our application that the own implementation of a socket-based data interface allows a tailored design of the interface featuring a good integration within the proprietary LabVIEW-based software. The developed TCP/IP-based data interface allows exchanging data with sampling rates of up to 100 ms between the simulation and the LabVIEW-based testbench software. We assess the sampling rates of the data interface to be sufficient for the usage in the presented HIL-system.

3.3 Hardware Level

The proposed HIL-system is implemented in a real testbench in the labs of WILO SE (see Figure 6). The hardware level includes a real pump which is installed in a hydraulic circuit of a testbench. The testbench includes facilities to measure the pressure difference across the pump and the volume flow rate in the hydraulic circuit. It includes an adjustable valve to emulate the hydraulic boundary conditions of the simulated building. Both the simulation in Dymola and the LabVIEW-based software to control the facilities for data acquisition run on a desktop PC placed near the testbench. We use Dymola with default settings and its capability to synchronize with an external Dynamic Data Exchange (DDE) server for (soft) real-time simulation on a Windows 7 (64-bit) platform. The LabVIEW-based software is used to operate the data acquisition and switch unit as well as providing a proprietary interface to the low-level pump motor control.

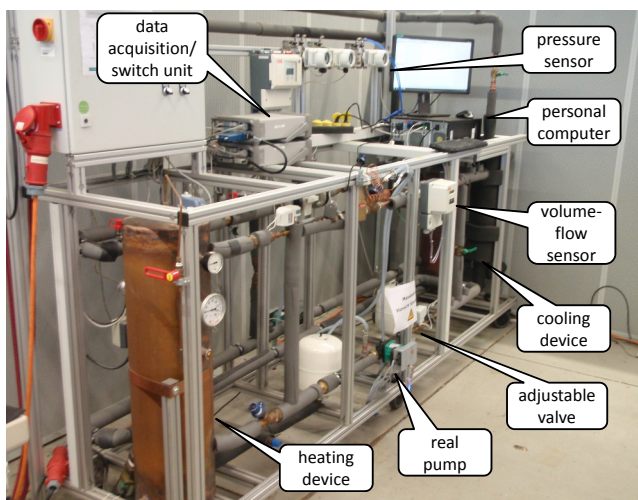


Figure 6. An implementation of the HIL-concept in a real testbench.

4 Results

To check the plausibility and evaluate the performance of the proposed HIL-setup for a circulating pump we present the results of a comparison between measured data from a HIL-simulation with calculated results of a full simulation in this section. Note, here *HIL-simulation* relates to a simulation of a building where the simulation is coupled to a real pump and *full simulation* relates to a simulation where the building is simulated including a model for a pump. In both cases we model a one family dwelling as described in section 3.1.

4.1 HIL-Simulation

In this section results for HIL-simulations are presented where a constant and a variable pressure difference control scheme are applied to control the pump. The set point of the pump head is set to 58.84 kPa and 21.083 kPa for the constant and variable pressure difference control scheme, respectively. In this specific test case the outside air temperature is set to -12 °C and no solar radiation is considered. These specific boundary conditions are used to calculate the maximum needed power of the heating system. We model occupant actions by a ventilation schedule which assumes that the air is exchanged in every room once in the morning and once in the evening for half an hour. The rooms are ventilated separately with an offset of half an hour. The average air exchange by infiltration is assumed to be 0.5 h⁻¹ over the whole day and during ventilation the thermostatic valves are closed. The set points of the room temperatures and basic parameters of the building model are reported in Table 1. A night control mode for the heating system is implemented which lowers the feed temperature of the heating medium between 10pm and 6am from 75 °C to 46 °C. The described boundary conditions apply for both control schemes.

Constant Pressure Difference Control Scheme

The results for a real-time HIL-simulation of a constant pressure difference control scheme for one day are presented in Figure 7. The measured volume flow rate and pump head in the hydraulic circuit, Q_m and H_m and the corresponding simulated values Q_s and H_s of the full simulation are depicted. For better understanding the deviations of the volume flow rates the results of the simulated room temperature T_{Room} and the set temperature T_{Set} for the thermostatic valve of an exemplary room in the lower floor are presented in Figure 9.

Due to the set boundary conditions the volume flow rates Q_m and Q_s of the building vary between 100 and 800 l/h during the day. Gradients of about 700 l/h² occur when the thermostatic valves are reopened after the ventilation phases from 9am to 10am and 6pm to 8pm (see Figure 7 and 9). Oscillations of the head H_m are caused by the PI-control yielding a feedback of the

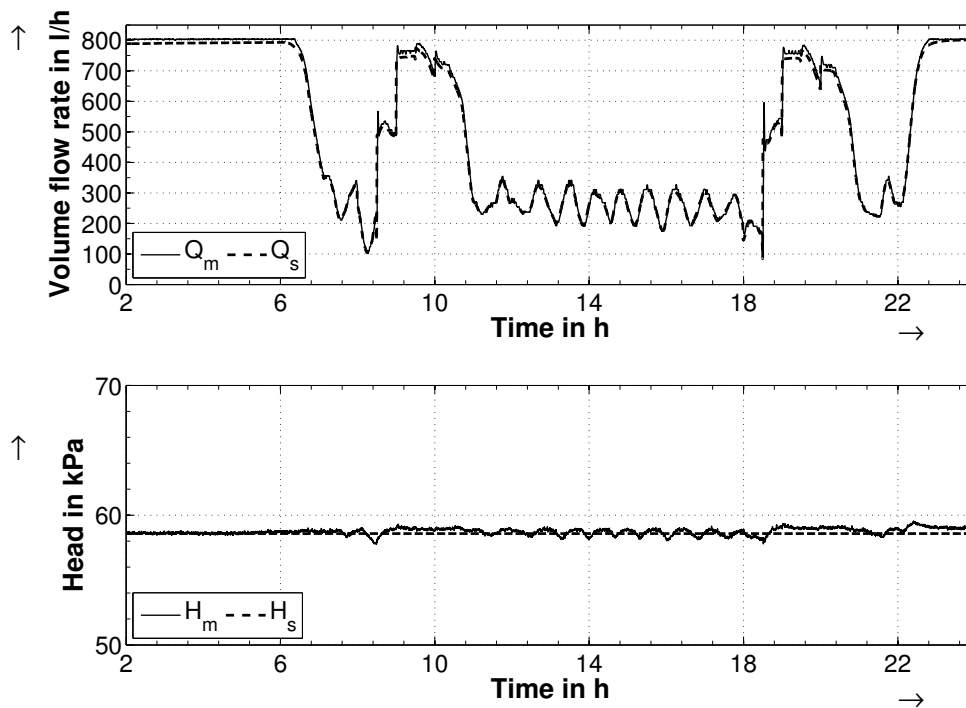


Figure 7. Comparison of a real-time HIL-simulation with full simulation for one day applying a constant pressure difference control scheme to the pump. Q_m , H_m measured values for HIL-simulation, Q_s , H_s simulated values for full simulation, volume flow rate and head of the pump, respectively, sampling rate: 1 s.

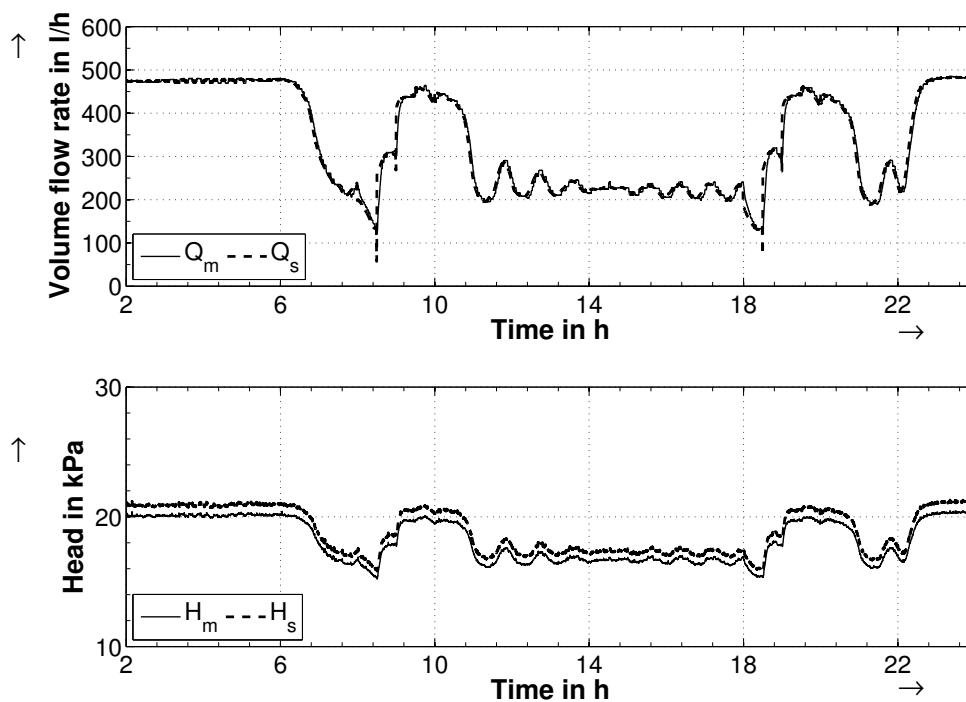


Figure 8. Comparison of a real-time HIL-simulation with full simulation for one day applying a variable pressure difference control scheme to the pump. Q_m , H_m measured values for HIL-simulation, Q_s , H_s simulated values for full simulation, volume flow rate and head of the pump, respectively, sampling rate: 1 s.

Table 1. Selected parameter of the building model of a one family dwelling according to energy saving ordinance from 1984. Design heat load calculated according to EN 12831 (CEN, 2003).

Parameter	Value
Total length	11.49 m
Total width	8.74 m
Total area	100.42 m ²
Room height	2.6 m
Number of floors	2
Infiltration rate	3 h ⁻¹
Ventilation rate	0.5 h ⁻¹
Set temperature bathroom	24 °C
Set temperature hallway	18 °C
Set temperature other rooms	22 °C
Design heat load	8.97 kW
Design volume flow rate	1.1 m ³ /h
Design head	22.56 kPa
Location	Mannheim/ Germany

volume flow rate oscillations to the pump head H_m . The results provide evidence that the volume flow rate control system faces no difficulties in emulating the simulated gradients. To quantify the quality of the emulation we compare the measured and simulated values by calculating a root mean squared error and a relative mean error of 17.79 l/h (2.40 %) and 307.92 Pa (0.44 %) for the volume flow rates and pump heads, respectively.

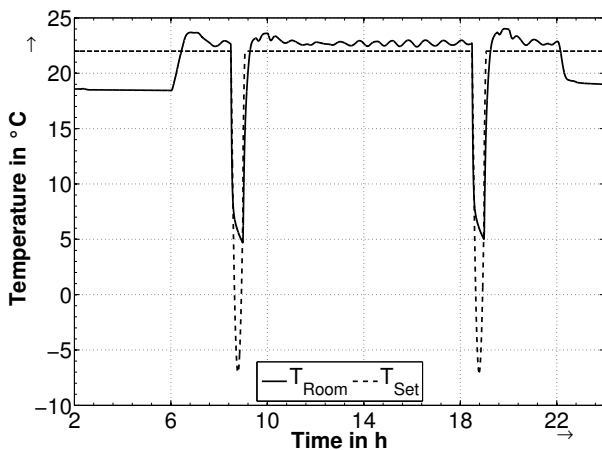


Figure 9. Simulated room temperature T_{Room} and set temperature T_{Set} of one room of the building for one day with constant pressure control scheme applied and sampling rate: 1 s.

Variable Pressure Difference Control Scheme

For a variable pressure difference control scheme we present the results of a one day HIL-simulation in Figure 8. Volume flow rates and pump heads Q_m and H_m refer to the measured values in the hydraulic circuit of

the testbench and Q_s and H_s to the simulated values for the full simulation, respectively. The corresponding results for the simulated room temperature T_{Room} and the set temperature T_{Set} for the thermostatic valve of an exemplary room in the lower floor are given in Figure 10. Temperature trajectories in Figure 9 and 10 are from the same room.

As the boundary conditions remain equal for the constant and the variable pressure difference control scheme, the trajectories of the volume flow rates Q_m and Q_s in Figure 7 and 8 show a similar pattern. Of interest are the pump head trajectories H_m and H_s , shown in Figure 8, which are lowered from set point 21.08 kPa to about 17.16 kPa from 11am and 6pm where the volume flow rate and respectively the heating demand is low compared to the ventilation times (see Figure 10). While maintaining the thermal comfort by keeping the room temperature T_{Room} above the set temperature T_{Set} during day time (Figure 10), the variable pressure difference control scheme reduces the pump head. This yields a decrease of the amount of energy needed to operate the pump.

To compare the measured and simulated values we calculate a root mean squared error and a relative mean error of 14.49 l/h (3.04 %) and 735.48 Pa (3.86 %) for the volume flow rates and pump heads, respectively.

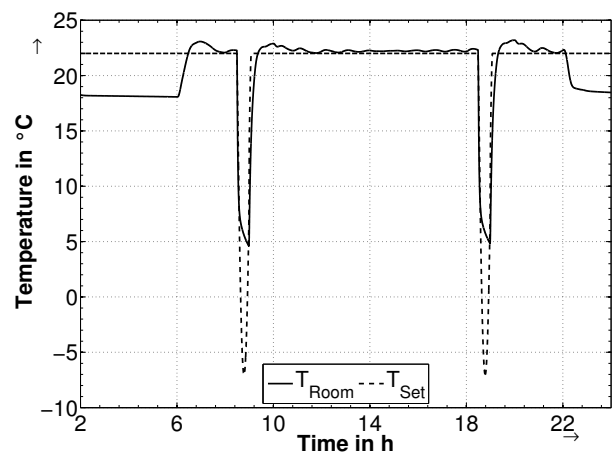


Figure 10. Simulated room temperature T_{Room} and set temperature T_{Set} of one room of the building for one day with variable pressure control scheme applied and sampling rate: 1 s.

4.2 Discussion

For HIL-simulation specific hardware exists which ensures simulation in hard real-time, e.g. by real-time capable operating systems. The HIL-simulations presented in this work have been conducted on a generic desktop PC running on Windows 7 using the described models (see section 3.1) without other than the reported adjustments. Our findings indicate that with sampling rates and feedback control in the range of seconds, soft real-time simulation is sufficient. This concurs with the findings

presented by Gäfvert et al. (2008). An implication of this is that for the presented application no additional investments in the simulation infrastructure are necessary for HIL.

We assume that increasing the complexity of the models, e.g. investigate multiple-family dwellings, will not cause any problems since solution times for one time step of the building model are well below the sampling time of the HIL-system. In our application the sampling time is limited to one second. This is related to the proprietary LabVIEW-based testbench software, mainly caused by an overhead generated from data handling algorithms in an integrated database.

The comparison of simulated results and measured data provide evidence that the solutions for the emulation of the hydraulic boundary conditions as well as the implemented data interface presented in this work show a sufficient performance for the investigated application with calculated mean relative errors less than 4 %.

Real hardware signals have a significant impact on the robustness of control algorithms. In Figure 7 simulated results for volume flow rate and pump head are continuous, whereas discrete data acquisition and measurement noise alter the measured signals. In contrast to the transition of model-based designed control algorithms which have not been tested on real hardware, HIL-simulation assisted transition offers the additional benefit to deduce improvements and increase the robustness of model-based designed algorithms as these signals are considered already during the design process and prior to commissioning. The quality of the emulation systems needs to be evaluated carefully to exclude an influence of these devices on the measurements.

HIL-simulation in the described context offers the benefit that, in contrast to measurements in real building energy and control systems, the boundary conditions of test scenarios can be chosen freely by the modeller and be reproduced almost equally on a testbench for multiple testing.

5 Conclusion

In this paper we present the application of the HIL-method to a building energy and control system to investigate circulating pump control. The presented work evaluates how HIL-simulation of building energy control systems using Modelica can be used to bridge the gap between design and commissioning stage of control algorithms for HVAC components, e.g. circulating pumps. We describe in detail the solutions found for the emulation of the hydraulic boundary conditions and a socket-based data interface. HIL-simulations are performed on an implementation of the concept. The quality of implementation is evaluated with results obtained from HIL-simulations where a constant and a variable pressure control scheme are applied to the pump. For a comparison

of measured data against simulated results we calculate a root mean squared error and a relative mean error of 17.79 l/h (2.40 %) and 307.92 Pa (0.44 %) for the volume flow rates and the pump heads applying a constant pressure difference control scheme, respectively. For results obtained by applying a variable pressure difference control scheme we calculate a root mean squared error and a relative mean error of 14.49 l/h (3.04 %) and 735.48 Pa (3.86 %) for the volume flow rates and the pump heads, respectively.

With the HIL-system presented in this paper, obstacles of the transition from design to commissioning stage of circulating pump control are solved at once:

- Model-based designed algorithms are tested directly on real hardware without reimplementing in an application controller;
- The usage of object-oriented model libraries allows to investigate different types of buildings and HVAC systems by exchanging component models on one hardware setup.

If correctly applied cost and time savings are expected for manufacturers as the methodology significantly speeds up and simplifies the transition process. Additionally, the methodology offers the benefit of enabling the feedback of insights to the design process from running newly developed control algorithms on real hardware.

A possibility to accelerate the still required reimplementing of the developed algorithms is using automated code generation techniques. HIL-simulation is an additional tool in the transition process. Nevertheless, to complete product development thorough testing and a quality insurance process are still necessary.

Results presented in this paper partly rely on proprietary code especially for the HVAC-system models. A future development could comprise of an open source library providing ready-to-use models for HIL of thermal and hydraulic components of building energy and control systems.

The methodology of this work has been tested with models for one family dwelling with a limited complexity. Future work could focus on expanding to more complex building models in terms of the number of modelling equations.

6 Acknowledgements

This research is part of a master thesis which took place at WILO SE supervised by the Institute for Energy Efficient Buildings and Indoor Climate. We would like to thank WILO SE for financial support of the research activities.

References

- ASHRAE, 2004. Standard Method of Test for the Evaluation of Building Energy Analysis Computer Programs, July 2004. American Society of Heating, Refrigerating and Air-Conditioning Engineers, Atlanta, USA.
- T. Bellmann. Interactive Simulations and Advanced Visualization with Modelica. In F. Casella, editor, *Proceedings of the 7th Modelica Conference*, pages 541–550. Linköping University Elektronic Press, Como, Italy, September 20–22 2009. doi:10.3384/ecp09430056.
- M. Bonvini, F. Donida, and A. Leva. Modelica as a Design Tool for Hardware-in-the-Loop Simulation. In F. Casella, editor, *Proceedings of the 7th International Modelica Conference*, pages 378–385. Linköping University Elektronic Press, Como, Italy, September 20–22 2009. doi:10.3384/ecp09430119.
- CEN, 2003. Heating Systems in Buildings - Method for Calculation of the Design Heat Load; German Version, DIN EN 12831, 2003. European Committee for Standardization (CEN), Brussels, Belgium.
- K. Chen, R. Streblov, D. Müller, A. Benigni, C. Molitor, and A. Monti. Hardware-in-the-Loop Simulationsverfahren für Hausenergiesysteme. In *Proceedings of the Fourth German-Austrian IBPSA Conference*. IBPSA-Germany, Berlin University of the Arts, September 2012.
- A. Constantin, R. Streblov, and D. Müller. The Modelica HouseModels Library: Presentation and Evaluation of a Room Model with the ASHRAE Standard 140. In H. Tummescheit and K.-E. Årzén, editors, *Proceedings of the 10th International Modelica Conference*, pages 293–299. Linköping University Elektronic Press, Lund, Sweden, March 10–12 2014. doi:10.3384/ecp14096.
- R. De Coninck, R. Baetens, D. Saelens, A. Woyte, and L. Helsen. Rule-Based Demand-Side Management of Domestic Hot Water Production With Heat Pumps in Zero Energy Neighbourhoods. *Journal of Building Performance Simulation*, 7(4):271–288, 2014. doi:10.1080/19401493.2013.801518.
- DWD, 2013. Test reference year, Deutscher Wetter Dienst (DWD), Offenbach, Germany, <http://www.dwd.de/TRY>, Accessed: 25.04.2015.
- Dymola, 2013. Dassault Systemes AB, Lund, Sweden, <http://www.3ds.com/products-services/catia/portfolio/dymola>.
- EU, 2009. Directive 2009/125/EC of the European Parliament and of the Council Establishing a Framework for the Setting of Eco-Design Requirements for Energy-Related Products, October 2009.
- M. Gäfvert, T. Skoglung, H. Tummescheit, J. Windahl, H. Wikander, and P. Reuterswärd. Real-Time HWIL Simulation of Liquid Food Process Lines. In B. Bachmann, editor, *Proceedings of the 6th International Modelica Conference*, volume 2, pages 709–715. The Modelica Association and University of Applied Sciences Bielefeld, Bielefeld, Germany, March 3–4 2008.
- D. Maclay. Simulation Gets Into the Loop. *IEEE Review*, 43(3):109–112, May 1997.
- Microsoft, 2015. Winsock library by Microsoft <http://msdn.microsoft.com/en-us/library/ms740673%28VS.85%29.aspx>, Accessed: 15.04.2015.
- D. Müller and A. Hosseini Badakhshani. Gekoppelte Gebäude- und Anlagensimulation mit Modelica. In *Proceedings of the 3rd German-Austrian IBPSA Conference*, September, 22–24, Vienna, Austria 2010.
- NI, 2015. LabVIEW, National Instruments, Austin, USA, <http://www.ni.com/labview/>.
- Solar-Computer, 2012. SOLAR-COMPUTER GmbH, Göttingen, Germany, <http://www.solar-computer.de>.
- M. Wetter. *Building Performance Simulation for Design and Operation*, chapter A View on Future Building System Modeling and Simulation, pages 1–28. Taylor & Francis, 2011a. ISBN 978-0-415-47414-6.
- M. Wetter. Co-Simulation of Building Energy and Control System with the Building Controls Virtual Test Bed. *Journal of Building Performance Simulation*, 4(3):185–203, 2011b. doi:10.1080/19401493.2010.518631.
- M. Wetter, W. Zuo, T. S. Nouidui, and X. Pang. Modelica Buildings Library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014. doi:10.1080/19401493.2013.765506.
- D. Winkler and C. Gühmann. Hardware-in-the-Loop Simulation of a Hybrid Electric Vehicle Using Modelica/ Dymola. In *Proceedings of the 22nd International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium and Exhibition*, pages 1054–1063, Yokohama, Japan, 2006.
- P. Xu, P. Haves, and J. Deringer. A Simulation-Based Testing and Training Environment for Building Controls. In *Proceedings of SimBuild 2004*, Boulder, USA, 2004.

Automatic GPU Code Generation of Modelica Functions

Hilding Elmqvist¹, Hans Olsson¹, Axel Goteman^{1,2}, Vilhelm Roxling^{1,2},
Dirk Zimmer³, Alexander Pollok³

¹Dassault Systemes, Lund, Sweden, {Hilding.Elmqvist, Hans.Olsson}@3ds.com

²Lund Institute of Technology, Lund, Sweden, {axel.goteman, vilhelm.roxling}@gmail.com

³Institute of System Dynamics and Control, DLR, Germany, {Dirk.Zimmer, Alexander.Pollok}@dlr.de

Abstract

Modelica users can and want to build more realistic and complex models. This typically means slower simulations. In the past, the speed of single CPUs has increased significantly to partly compensate, but more recently, there has been a shift to multi-core architectures. This is taken to the extreme in Graphics Processing Units (GPUs).

This paper discusses code generation for GPU cores. This is important when the model has regular structure, for example, discretization of PDEs. The behavior of each cell can then be partly described by a function call. The evaluation of such calls can then be made in parallel on the GPU cores. The same function is thus executed on every GPU core, but operates on different data; the data of its cell.

Our GPU code generator automatically generates code for Modelica functions, i.e. no additional language constructs are needed. The function is just annotated as suitable for execution on a GPU.

Keywords: Modelica functions, Multi-core, GPU, CFD

1 Introduction

Modelica users can and want to build more realistic and complex models. This typically means slower simulations. The speed of CPUs has of course increased enormously to partly compensate. But now it's important to utilize the many cores available in modern computer architectures.

The paper (Elmqvist, et al., 2014) presents an algorithm for automatic partitioning of model equations onto CPU cores. This technique is now available in Dymola 2016 (Dassault Systemes, 2015).

This paper discusses code generation for GPU (Graphics Processing Unit) cores. This is important when the model has regular structure, for example, discretization of PDEs. The behavior of each cell can then be partly described by a function call. The evaluation of such calls can then be made in parallel on the GPU cores. The same function is thus executed on

every GPU core, but operates on different data; the data of its cell.

We believe GPU code generation should be transparent for the user. The user only needs to give a hint that a certain function is suitable for GPU execution. In addition to the simplification for the user, it enables better portability of Modelica code, since if a tool does not support GPU code generation, it can simply ignore the annotation. The drawback might be that the user does not have full control of how the GPU and its memory are utilized; i.e. might not be able to get optimal speed.

Another important advantage with automatic GPU code generation is that built-in operators such as matrix multiplication and overloaded operators can also benefit. In addition, it allows reuse of normal Modelica library functions and automatically generating GPU code for them.

(Gebremedhin, et al., 2012) proposes an extension to Modelica called ParModelica, which introduces special **kernel function** and **parallel function** declarations and special variable declaration prefixes to indicate what memory to use for the variables: **parglobal**, **parlocal**, etc.

The outline of this paper is as follows. First a general introduction to GPU architecture and programming is given. Then follows principles of automatic GPU code generation from Modelica functions. Finally several examples are presented.

The speed-up factor varies for the different problem formulations. The best speed-up so far for this early Dymola prototype is about 5 times. It should be noted that this was achieved on a laptop with NVIDIA's Quadro K2100M GPU chip and an Intel Core i7-4800 MQ processor.

2 GPU Architectures and Programming Models

In this section, a short introduction to GPU architectures is made, bringing up some of the fundamentals and the aspects that are considered most important for this paper, concerning performance.

After that follows a short introduction to CUDA, the programming model used in this project, where some more performance considerations are brought up, ending with a hands-on example and some practical details for the code generation. Some of the aspects are directed to the user, to get an idea of the kind of code that could be accelerated by the GPU, and some aspects are rather for the auto generation.

A thorough introduction to the topics is made by (Kirk, 2013) and (Goteman, et al., 2015).

2.1 GPU Architectures

The most fundamental difference between CPU's and GPU's, is that the CPU's are general purpose processors, designed to perform well on sequential code, whereas a GPU is designed to process massively parallel tasks, only. The large cache memories and control logics of a CPU are not provided for the cores of a GPU. Instead, the manufacturers focus on putting as many cores as possible on the chip, letting the threads share cache and control logic between them. A GPU today may have thousands of cores. That is possible, since GPU's only have to work on SIMD (single instruction, multiple data) instructions. That means that, when feeding the GPU a task, that task is the same for every thread. The only difference is which data the task is to be performed on. Now, a task is often a sequence of instructions that, as we will come to later, in code is expressed as a function. This function may contain e.g. an *if*-statement that causes divergence in terms of execution among the threads. The reason that threads may take different paths at such branching points is that the conditions may depend on thread specific data (such as position in a grid, temperature at that position, etc.). We'll come back to this in a bit.

A GPU is designed to process a large amount of threads as fast as possible, and it is important to see the distinction between that and to process every single thread as fast as possible. The GPU is not supposed to work on all threads at once, which generally is not possible, as the number of cores is too low for it. It is designed to have multiple times more threads loaded into registers, than it can execute. This allows for efficient *thread scheduling*, which means that whenever a thread is idle because of a long latency operation, such as a global memory access, the cores can switch to work on other threads that are ready and waiting for execution.

GPU's are good at thread scheduling, but all time spent on long latency operations cannot be hidden. To fully utilize a GPU, you'll want to let the cores work with floating point operations as much as possible. And even if today's chips have a bandwidth to global memory (RAM) of more than 200 GB/s, global memory accesses has to be considered for good performance. A good way to analyze this is to consider the number of floating point operations per global

memory access for a thread, often called the CGMA (Compute to Global Memory Access) ratio. It should obviously be kept as high as possible. If it is too low, there is no way to keep the cores busy, independently how the threads are scheduled. That is because, if the limited amount of data that can be delivered to the cores per time unit is lower than the rate at which the operations on the data can be executed, the data transferring has become a limiting factor. So already on a high level, as a user, it can be advantageous to have the CGMA ratio-thinking in mind when considering letting a function be computed on the GPU.

The threads are partitioned on many levels, and this partitioning can differ between different hardware architectures. But most architecture has a lowest partitioning level, at which the parts are called *warps*. On warp level, no divisions between threads are made. That means that if one thread in a warp has a long latency operation (or just any operation, for that matter) in an *if*-statement, but not the others, all threads in the warp will have to wait for that one thread. But the waiting is at least limited to the warp, which on most current architectures consists of 32 threads. That means that having *if*-statements does not necessarily mean a big difference in performance. The divergence can be organized to be minimized within warps, but as it can be hard to know how warps are arranged, and where the divergence will appear, divergent code should generally be avoided.

The last, and most important, aspect to bring up, is the process of transferring data between the CPU memory and the GPU memory. It is the main bottleneck of a GPU. The transfer speed is relatively low, and the transfer is often related to a lot of overhead work. That implies that there is no point to send work to the GPU, unless there is a lot of it. So if it is possible to avoid memory transfers of this kind, e.g. by not repeatedly transferring constant data, it should be done.

2.2 The NVIDIA CUDA Programming Model

CUDA is a parallel computing platform and programming model invented by NVIDIA. In this project, the extension CUDA C/C++ has been used, making it possible to program CUDA enabled GPU's in a C/C++ environment, with a few extensions.

In CUDA, a function that should be executed, or *launched*, in parallel is called a *kernel*. The kernel is launched for a number of threads, which are divided into *blocks*, creating a *grid* of blocks. The blocks are the level on which threads are loaded to registers for execution, meaning that when a block is loaded, it will not unload before all its threads are executed. Thus threads can only be synchronized within a block. Synchronization here means putting points in the code where the threads should wait and synchronize with

other threads before continuing execution. Recall that all threads are not executed at once.

Because all the threads in a block are loaded and unloaded into registers at once, there are limitations on the number of threads in a block. E.g. NVIDIA Quadro K2100M, the chip used for most experiments in this project, has a maximum block size of 1024 threads. The blocks are executed on *Streaming Multiprocessors* (SM's), and an SM can usually accommodate a number of blocks. It is on the SM level that warps are scheduled for execution, and it is therefore important to load as many threads as possible to each SM. On K2100M, each SM can have a maximum of 2048 threads loaded at once, and there are three SM's, giving a total of 6144 thread slots. So in order to fully utilize this chip, considering the scheduling of warps, at least 6144 threads should be launched. It may be interesting to know that each SM has 192 cores, giving a total of 576 cores, which equals the number of threads that can actually execute in parallel.

If all 6144 slots for threads are loaded during an entire kernel execution, the kernel is said to have 100 % *occupancy*. Of course this rarely happens since some blocks are bound to finish sooner than others. For good performance, the occupancy should be kept as high as possible, to allow for as much warp scheduling as possible. If a kernel on K2100M would be launched with blocks of 768 threads, the SM's would still only have place for two whole blocks, resulting in a lower occupancy. Or if a kernel is complex, each thread may require more registers, forcing down the number of threads loaded into an SM, thus decreasing the occupancy. There are more aspects that can affect the occupancy, and it is definitely a term that is good to know when considering GPU performance in general.

2.2.1 Example: vector addition

It may be of interest to see how all this could look in practice. First a few notes about CUDA C/C++:

- 1.) Generally, when inside a kernel, i.e. when code is executed on the GPU, no data that is not allocated on GPU memory can be accessed.
- 2.) Built-in primitives such as *int* and *float*, pointers, and structs can be copied to the GPU as arguments to the kernel (without deep copy). Large sets of data, like arrays, have to be copied using some CUDA API function.
- 3.) A kernel's return type must be of type void.
- 4.) A kernel may call other functions on the GPU, called *device* functions.
- 5.) Only a subset of C/C++ is supported.
- 6.) Thrust is a C++ STL based library for CUDA.

Two arrays can be added on the CPU in the following function:

```
void vectorAddCPU(const double *a, size_t n,
                 const double *b, double *c){
    for(size_t i=0; i<n; ++i){
```

```
        c[i] = a[i]+b[i];
    }
}
```

It is clear that this is a very parallel task. n threads could be launched, where each thread should have an individual variable i in some way. The simplest way to recognize that something is parallelizable is usually when it is placed in a for-loop, or in nested for-loops, and it does not depend on previous iterations. Below is a kernel for vector addition:

```
__global__
void vectorAddGPU_kernel(const double *a, size_t n,
                        const double *b, double *c){
    size_t i=threadIdx.x+ blockIdx.x*blockDim.x;
    if(i<n){
        c[i]=a[i]+b[i];
    }
}
```

Note the keyword `__global__` needed before the return type. First the thread is identifying itself using the thread specific variable `threadIdx`, the variable `blockDim`, and the block specific variable `blockIdx`. Those are variables of the simple type *dim3*, having three members: x , y and z . This helps to arrange threads according to your problem in up to three dimensions. In this case the problem is obviously one dimensional. The *if*-statement is needed to prevent memory access violations in cases where more threads are launched than needed. That is usually the case when many blocks are launched, since all blocks have the same size.

However, some operations are needed to call the kernel:

```
void vectorAddGPU(const double *a, size_t n, const
double *b, double *c){
    // Allocate GPU memory.
    double *a_d, *b_d, *c_d;
    cudaMalloc(&a_d, n*sizeof(double));
    cudaMalloc(&b_d, n*sizeof(double));
    cudaMalloc(&c_d, n*sizeof(double));

    // Copy a and b to GPU.
    cudaMemcpy(a_d, a, n*sizeof(double),
               cudaMemcpyHostToDevice);
    cudaMemcpy(b_d, b, n*sizeof(double),
               cudaMemcpyHostToDevice);

    // Define grid and block dimensions
    dim3 block = dim3(1024,1,1);
    dim3 grid = dim3((n+1023)/1024,1,1);

    // Launch kernel
    vectorAddGPU_kernel
```

```

<<<grid,block>>>(a_d, n, b_d, c_d);

// Copy result back to the CPU.
cudaMemcpy(c, c_d, n*sizeof(double),
  cudaMemcpyDeviceToHost);

// Free GPU memory
cudaFree(a_d);
cudaFree(b_d);
cudaFree(c_d);
}

```

First memory is allocated on the GPU for the three vectors, using the CUDA API function `cudaMalloc()`. Then a and b are copied to the allocated memory, using the CUDA API function `cudaMemcpy()`. Then the blocks and the grid are defined, and the kernel is launched. Note the CUDA syntax to specify the kernel launch settings. When the addition is completed on the GPU, the result in c has to be copied back, again using `cudaMemcpy()`. Last of all the GPU memory has to be freed with the CUDA API function `cudaFree()`.

3 GPU Code Generation

The basis for our code generator for Modelica functions is that we recognize that specially marked functions (`annotation(gpuFunction=true)`) satisfy certain for-loop patterns, and for those functions automatically generate GPU-code. The GPU-code consist of a wrapper that allocates variables, copies default values, executes the main body (automatically calling generated CUDA kernel-functions), and then copy back outputs. Any functions called in a kernel function are mapped to device functions.

3.1 Variable allocations

All array variables of the function must either have unknown size (only allowed for inputs), or a size given as a simple arithmetic function of other sizes and integer literals. The unknown array sizes are also propagated to the kernel function. (For performance reasons – and to catch errors – it is good to have as few unknown sizes as possible.) The arrays are allocated on the GPU and existing values copied to the GPU; this allows non-input variables to be assigned a default value in a binding expression. Protected arrays are treated as outputs of the kernel function.

3.2 Loop patterns

The first pattern for the algorithm is that the entire algorithm is a (possibly nested) for-loop with range `1:size(array, literal)` and inside the loop any algorithmic code satisfying certain assumptions. The code inside the for loop(s) is mapped to a kernel function; and the (nested) for-loop(s) are replaced by a parallel launch of the kernel function – and checking the index in the kernel function. As an example, consider the function:

```

function vectorAdd
input Real a[:];
input Real b[size(a,1)];
output Real c[size(a,1)];
algorithm
  for i in 1:size(a,1) loop
    // Kernel part
    c[i]:=a[i]+b[i];
  end for;
annotation(gpuFunction=true);
end vectorAdd;

```

This Modelica function is translated to the previously given `vectorAddGPU` and `vectorAddGPU_kernel` code.

3.3 Time integration on the GPU

The second pattern (intended to handle time-integration on the GPU) is a for-loop (with arbitrary index) that contains one or more instances of the first pattern. Each instance of the first pattern is then mapped to a kernel function and called at the appropriate place. The rest of the body may contain assignments; and any array assignment is mapped to a device-to-device copy. The main benefit of this pattern is that we do not need to copy back outputs from the GPU until the end of the function, and device-to-device copy is normally a lot faster.

As an example, consider the following partial differential equation with $v(0,0)=0$:

$$\frac{\partial v(x, t)}{\partial x} = F(v(x, t))$$

$$\frac{\partial v(x, t)}{\partial t} = v(x, t)$$

One way of solving such PDEs is to discretize in the x direction and use an ODE solver for the resulting equations. However, fine grained spatial discretization requires short time increments and it might then be better to use a fixed step size Euler method for time integration.

The function `IntegrateF` below implements such a solution.

```

function IntegrateF
input Real v[:];
output Real next_v[size(v,1)]:=v;
input Real dt;
input Integer nSteps;
protected
  Real temp_v[size(v,1)];
algorithm
  // Loop in wrapper-code
  for step in 1:nSteps loop
    // Handled using device-to-device copy:
    temp_v:=next_v;
    for i in 1:size(v, 1) loop

```

```

// Kernel part
next_v[i]:=temp_v[i]+dt*(if i>1 then
  F(temp_v[i-1]) else 0);
end for;
end for;
end IntegrateF;

```

For the function IntegrateF the main code is similar to vectorAddGPU. The difference is that the called GPU function part is replaced by a loop as follows:

```

dim3 block0=dim3(1024, 1, 1);
dim3 grid0=dim3((x0_0dim0+1023)/1024,1,1);
{
  int st;
  for(st=1; st<=nSteps; st+=1) {
    cudaMemcpy(tem_vGPU, next_vGPU,
      temp_vGPUs*sizeof(double),
      cudaMemcpyDeviceToDevice);
    IntegrateFcuda<<<grid0,block0>>>(
      vGPU, vdim0,next_vGPU,
      dt,nSteps, temp_vGPU);
  }
}

```

i.e. the time integration loop is executed on the CPU. There is synchronization between each iteration, i.e. all launched kernel calls must have completed. Note that the statement `temp_v:=next_v` is translated to a copy call on the GPU memory.

The copying can be avoided by swapping arguments to the kernel calls. It is possible to manually avoid it (note: `next_v` is initialized to `v`) – assuming an even number of steps:

algorithm

```

// Loop in wrapper-code
for step in 1:2:nSteps loop
  for i in 1:size(v, 1) loop
    // Kernel part: first kernel function
    temp_v[i]:=next_v[i]+dt*(if i>1 then
      F(next_v[i-1]) else 0);
    end for;
  for i in 1:size(v, 1) loop
    // Kernel part: second kernel function
    next_v[i]:=temp_v[i]+dt*(if i>1 then
      F(temp_v[i-1]) else 0);
    end for;
  end for;
end for;

```

Automating the entire generation of time integration code from the model code would be a possibility for the future, by using synchronous partitions and specifying a solver method associated with the clock.

3.4 For-expressions

An alternative pattern to nested for-loops would be arrays assigned in for-expressions; it would simplify some of the assumptions below, but for performance

reasons we would likely need to fuse the loops from multiple for-expressions.

3.5 Assumptions for kernel code

The assumptions on the inner code are (these could be automatically verified, but this is not yet included in the prototype):

- All array indices are valid; based on the array sizes.
- Any right-hand-side variable is not assigned in the inner code. (An exception can be made for scalar temporaries that are initialized in the inner code.) This explains why we need two arrays `next_v` and `temp_v` in the example above.
- Each left-hand-side array element is only assigned once.
- Currently only access to scalar variables, and scalar element of arrays in the right hand side, i.e. slices are not supported.

4 Application examples

4.1 Matrix Operations

(Gebremedhin, et al., 2012) uses matrix multiplication as one bench mark example for an extension to Modelica called ParModelica which introduces special **kernel function** and **parallel function** declarations and special variable declaration prefixes to indicate what memory to use for the variables: **parglobal**, **parlocal**, etc.

In our approach, such a matrix multiplication function can be coded in Modelica as follows. Note that the only new element compared to Modelica version 3.3 (Modelica, 2014) is the annotation.

```

function Multiply
  input Real A[:,:];
  input Real B[size(A,2),:];
  output Real C[size(A,1),size(B,2)];
protected
  Real temp;
algorithm
  for i in 1:size(A,1) loop
    for j in 1:size(B,2) loop
      temp := 0;
      for k in 1:size(A,2) loop
        temp := temp + A[i, k]*B[k, j];
      end for;
      C[i, j] := temp;
    end for;
  end for;
  annotation(gpuFunction=true);
end Multiply;

```

It is translated to two functions. The kernel function which runs on the GPU is shown below:

```

#include <stddef.h>
__global__
void Multiply_cuda(
  double const * A, size_t Adim0,
  size_t Adim1,
  double const * B, size_t Bdim1,
  double * C)
{
  double temp;
  temp=0;
  int i = 1+threadIdx.x +
    blockDim.x*blockIdx.x;
  int j = 1+threadIdx.y +
    blockDim.y*blockIdx.y;
  if ((i<=Adim0) && (j<=Bdim1)) {
    temp = 0;
    {
      int end_ = Adim1;
      int k;
      for(k = 1; k <= end_; k += 1) {
        temp = temp +
          A[(i-1)*Adim1+(k-1)] *
          B[(k-1)*Bdim1+(j-1)];
      }
      C[(i-1)*Bdim1+(j-1)] = temp;
    }
  }
  return;
}

```

```

BGPUS*sizeof(double));}

if (CGPU&&(CGPUS<CGPUs))
  {CGPUS=0;cudaFree(CGPU);CGPU=0;}
if (!CGPU)
  {CGPUS=CGPUs;cudaMalloc((void**)&CGPU,
    CGPUS*sizeof(double));}

/* GPU Memory copy to */
cudaMemcpy(AGPU, A,AGPUs*sizeof(double),
  cudaMemcpyHostToDevice);
cudaMemcpy(BGPU, B,BGPUs*sizeof(double),
  cudaMemcpyHostToDevice);
cudaMemcpy(CGPU, C,CGPUs*sizeof(double),
  cudaMemcpyHostToDevice);
/* Call GPU function */
dim3 block=dim3(32, 32, 1);
dim3 grid=dim3((Adim0+31)/32,
  (Bdim1+31)/32, 1);
GPUfunction_cuda<<<grid,block>>>(AGPU,
  Adim0, Adim1,BGPU, Bdim1,CGPU);
/* GPU Memory copy from */
cudaMemcpy(C, CGPU,CGPUs*sizeof(double),
  cudaMemcpyDeviceToHost);
}

```

The other function runs on the CPU to allocate memory for the GPU, copy data to and from the GPU and to invoke the kernel function:

```

extern "C"
void Multiply(
  double const * A, size_t Adim0,
  size_t Adim1,
  double const * B, size_t Bdim1,
  double * C)
{
  /* GPU Memory declaration */
  static double * AGPU=0;
  static size_t AGPUS=0;
  size_t AGPUs;
  static double * BGPU=0;
  static size_t BGPUS=0;
  size_t BGPUs;
  static double * CGPU=0;
  static size_t CGPUS=0;
  size_t CGPUs;
  /* GPU Memory size */
  AGPUs=Adim0*Adim1;
  BGPUs=Adim1*Bdim1;
  CGPUs=Adim0*Bdim1;

  /* GPU Memory allocation */
  if (AGPU&&(AGPUS<AGPUs))
    {AGPUS=0; cudaFree(AGPU); AGPU=0;}
  if (!AGPU)
    {AGPUS=AGPUs;cudaMalloc((void**)&AGPU,
      AGPUS*sizeof(double));}

  if (BGPU&&(BGPUS<BGPUs))
    {BGPUS=0; cudaFree(BGPU); BGPU=0;}
  if (!BGPU)
    {BGPUS=BGPUs;cudaMalloc((void**)&BGPU,

```

4.1.1 Timing

Timing of the function `Multiply` for matrix multiplication was done for different sizes (n) of square matrices. Table 1 summarizes the execution times on CPU and on GPU and the speedup factor.

Table 1: GPU Speed-up for matrix multiplication. The speedup values show the CPU/GPU time ratio.

N	CPU[s]	GPU[s]	Speedup
50	0.000453	0.000438	1.03
100	0.00362	0.0018	2.01
200	0.0275	0.00996	2.76
500	0.506	0.142	3.56
1000	6.37	1.11	5.74

Note that the CPU-performance for large matrices is sensitive to caches; which for n being a power of 2 can increase the CPU time by up to a factor of 3; the chosen dimensions avoid that effect.

4.2 Cold Plate

As a second application example, a cold-plate is modeled. These are, for instance, used to cool power-electronics. The dissipated heat is transported away from the source by conduction and convection. In this two-dimensional example, a single fluid pipe is surrounded by two rectangular conducting plates. For the sake of clarity, we created a simple monolithic model that can mentally be split up into three kinds of cells: thermal conduction cells, fluid volume cells, and fluid flow cells. This is illustrated in Figure 1.

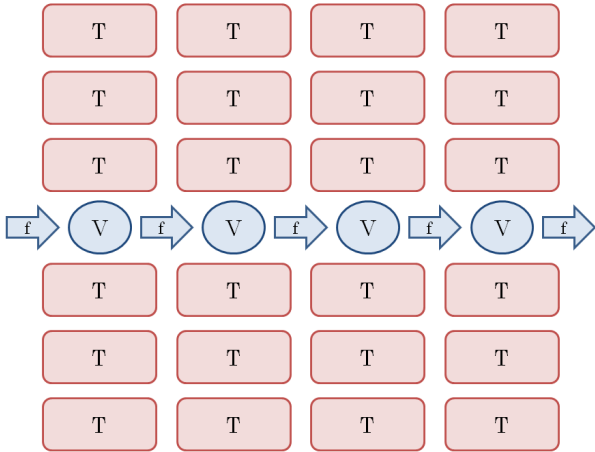


Figure 1: Illustration of a simple cold plate model, split up into thermal (T), fluid volume (V), and fluid flow (f)-cells.

Overall, the behavior of the cells is similar to models in the `Modelica.Fluid` and `Modelica.Thermal.HeatTransfer` domains. The number of cells in both dimensions is configurable, tuning the number of variables and states. In the thermal conduction cells, heat is stored and conducted to the four neighboring thermal cells or fluid volume cells. This is a slight simplification, as the resulting dynamic is anisotropic. In the fluid volume cells, balance equations for mass and energy are established.

Fluid is transported between the volume cells by flow-cells. These calculate the mass flow based on the pressure values of the neighboring volume cells by the function `Modelica.Fluid.Pipes.BaseClasses.WallFriction.Detailed.massFlowRate_dp()`. This calculation is quite involved and is therefore done in parallel by the GPU as shown in the following Modelica code, i.e. since the kernel function calls `massFlowRate_dp`, CUDA code is generated as a device function.

```
function WallFriction
  input Real dp[::];
  input Real d[size(dp,1)-1];
  input Real da;
  input Real db;
  input Real v[size(dp,1)-1];
  input Real va;
  input Real vb;
  input Real celllength;
  input Real diameter;
  input Real roughness;
  input Real m_flow_small;
  output Real m_flow[size(dp,1)];
algorithm
  for i in 1:size(dp,1) loop
    m_flow[i] := if i == 1 then
      massFlowRate_dp(dp[1], da, d[1],
        va, v[1], celllength, diameter, roughness, m_flow_small)
    else if i == size(dp,1) then
      massFlowRate_dp(dp[i], d[i-1], db,
```

```
  v[nX], vb, celllength, diameter, roughness, m_flow_small)
    else
      massFlowRate_dp(dp[i], d[i-1], d[i],
        v[i-1], v[i], celllength, diameter, roughness, m_flow_small);
    end for;
  annotation(gpuFunction=true);
end WallFriction;
```

As initial conditions, the temperature of all cells was set to 295K. A constant pressure gradient of 0.01bar was applied and the inlet temperature was set to 373.15K, resulting in a heating transient. Since the fluid transport is rather stiff, a small step-size has to be applied when using the RK2 method for integration.

Table 2: GPU Timing [s] and speed-up for cold plate model

nx	ny	CPU	GPU	CPU/GPU
256	256	17.2	12.1	1.42
512	512	91.7	42.9	2.13
500	200	26.0	18.1	1.43
1000	100	28.6	20.2	1.41
2000	200	103.0	65.2	1.57

As a result of the parallelization, a speed-up by a factor of 2 was achieved in one case. Note, that the step size and simulated time are different for the different grids.

The presented example only showed a very simple model of a cold plate with a straight flow of cooling liquid. Nevertheless, we think that the measured performance gains can be roughly transferred to more complex designs.

4.3 Shallow Water

For wave power plants or off-shore constructions such as wind-turbines and oil-platforms, as well as for free floating objects such as ships, the interaction with the water surface is a key component for system simulations. For this purpose, the shallow water equations represent a set of partial differential equations (PDEs) that enable an efficient approximation of the surface dynamics (Vreugdenhil, 1994). The PDE for a 2D-surface in its simplest form is shown below:

$$\frac{\partial h}{\partial t} = -H \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} \right)$$

$$\frac{\partial v_x}{\partial t} = -g \frac{\partial h}{\partial x}$$

$$\frac{\partial v_y}{\partial t} = -g \frac{\partial h}{\partial y}$$

In this model, the velocity of the water flowing within the 2D surface is described by v_x and v_y . A gradient

(i.e. a difference between inflow and outflow) then causes the surface height h to raise or fall. Spatial gradients in the surface height then cause a counteracting acceleration of the water flow. H and g are parameters of the model. H is chosen as $L/4$, where L is the length of the side of the surface area, and g is set to 9.81 m/s^2 to model gravity realistically.

The PDE is transformed into an ODE by discretizing the space using finite differences and a staggered grid for the velocity and surface height, as depicted in Figure 2. The resolution of this grid can be set by using the parameter n .

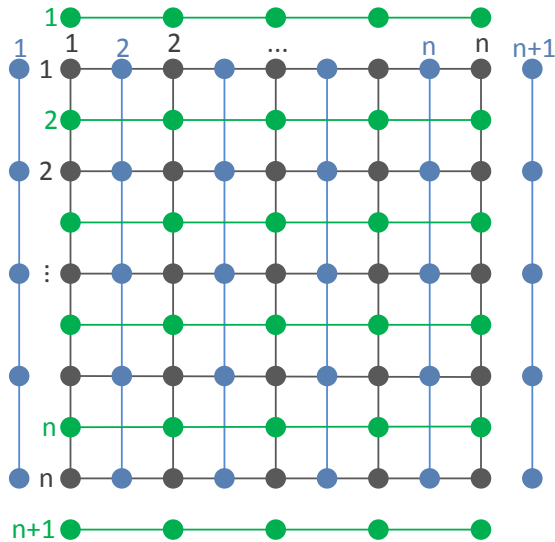


Figure 2: A staggered grid: black points symbolize the height grid h . Blue represents v_x and green v_y . For a size of n , the grid contains in total $3n^2+2n$ points.

Using this discretization, the computation can be written in form of a GPU function that performs a loop over the staggered grid:

```
function ShallowWater
  input Real h[:,:];
  input Real vx[size(h, 1) + 1, size(h, 2)];
  input Real vy[size(h, 1), size(h, 2) + 1];
  input Real dx;
  input Real L;
  input Real g;
  output Real der_h[size(h, 1), size(h, 2)];
  output Real der_vx[size(vx, 1), size(vx, 2)];
  output Real der_vy[size(vy, 1), size(vy, 2)];
protected
  Real H=L/4;
algorithm
  for iy in 1:size(h, 2) loop
    for ix in 1:size(h, 1) loop
      der_h[ix, iy] := H*(vx[ix, iy] - vx[ix + 1, iy] +
        vy[ix, iy] - vy[ix, iy + 1])/dx;
      der_vx[ix, iy] := if ix > 1 then
        g*(h[ix - 1, iy] - h[ix, iy])/dx else 0;
      der_vy[ix, iy] := if iy > 1 then
        g*(h[ix, iy - 1] - h[ix, iy])/dx else 0;
    end for;
  end for;
```

```
end for;
annotation(gpuFunction=true);
end ShallowWater;
```

This Modelica function is then included in a complete Modelica model. This model also describes the boundary conditions (closed boundary with zero velocity) and the initial state (zero velocity with a surface height that forms a Gaussian bell curve in the center). In addition, the model generates data for visualization.

The model can be simulated using, for example, a Runge-Kutta method of second order (RK2) with fixed step size of 40ms. Figure 3 shows the simulation result for $n=64$ after the Gaussian bell has “dropped” and created a typical circular wave front that grows until it is reflected at the boundaries.

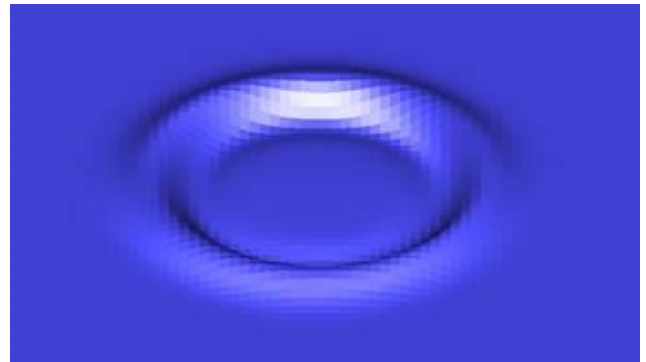


Figure 3: Dymola animation of a circular wave front in the shallow water model.

Given its large number of states, the model is well-suited for parallel computation on a GPU. On the other side, the actual computation of an element is relatively cheap in comparison to the required communication overhead with its neighbor cells.

To improve performance, the time integration was performed directly on the GPU, using the explicit Euler method (using code similar to the previously described function integrateF). The states could then be sampled from the GPU memory at a desired rate. Table 3 presents the results of our performance measurements. For larger models it becomes greater than a factor of 5. In the table, $nSteps$ is the number of iterations between each sample, and the results shows how important it can be to avoid unnecessary copying.

Table 3: GPU Speed-up for the shallow water simulation, using inlined integration. The values show the CPU/GPU time ratio.

nSteps\n	32	64	128	256
1	0.36	0.77	0.91	1.00
10	0.46	0.94	1.19	1.42
100	0.77	1.89	3.03	3.19
1000	1.01	3.06	5.30	5.12

The values in the table show the CPU/GPU time ratio of the simulations. n is the number of cells in one dimension. The CPU simulations are made by the same Modelica code, doing inline integration on the CPU instead.

Using GPU parallelization, PDEs for shallow water simulation can be better performed in combination with classic system simulation. In this way, the practical application range of Modelica can be extended.

5 Conclusions

Modelica models are getting more and more complex which means that simulations must be performed more efficiently. This paper demonstrates a technique to generate GPU code for Modelica functions in order to speed-up simulations by parallel execution on many GPU cores. No Modelica extensions are needed, only an annotation indicating that a certain function might be suited for execution on the GPU.

In our prototype implementation, and using the GPU of a laptop, a speed-up of 5 could be achieved in some cases.

Acknowledgements

This work has partly been performed as a master thesis project at Lund Institute of Technology. The first author served as an industrial advisor and Michael Doggett as the formal supervisor.

This paper is partly based on research performed within the ITEA2 project MODRIO. Partial financial support of the Swedish VINNOVA is highly appreciated.

Helpful discussions with Sven Erik Mattsson are appreciated. The cold plate model originated from input of Daniel Bender.

References

- Dassault Systèmes (2015): Dymola 2016. <http://www.Dymola.com>
- Elmqvist H., Mattsson S.E., Olsson H. (2014): Parallel Model Execution on Many Cores. Proceedings of the 10th International Modelica Conference March 10-12, 2014, Lund, Sweden.
- Gebremedhin M., Hemmati Moghadam A., Fritzon F., Stavåker K. (2012): A Data-Parallel Algorithmic Modelica Extension for Efficient Execution on Multi-Core Platforms. Proceedings 9th Modelica Conference, Munich, Germany, September 3-5, pp. 393-404. Download: <http://www.ep.liu.se/ecp/076/041/ecp12076041.pdf>
- Goteman A., Roxling V. (2015): GPU Usage for Parallel Functions and Contacts in Modelica, master's thesis Lund Institute of Technology, Lund, Sweden. (To be published)
- Kirk D.B., Hwu W. (2013): Programming Massively Parallel Processors, 2nd edition.

Modelica (2014): Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.3, Revision 1, June 11, 2014.

<https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf>

Vreugdenhil C.B. (1994), Numerical Methods for Shallow-Water Flow, Kluwer Academic Publishers, ISBN 0792331648

Constructs for Meta Properties Modeling in Modelica

Hilding Elmqvist¹, Hans Olsson¹, Martin Otter²

¹Dassault Systemes, Sweden, {Hilding.Elmqvist, Hans.Olsson}@3ds.com

²Institute of System Dynamics and Control, DLR, Germany, Martin.Otter@dlr.de

Abstract

This article proposes two new language constructs for meta-properties modeling in Modelica: (1) Accessing all instances of a given class and (2) extracting in a convenient way the desired information from such instances by allowing to pass type compatible model instances as arguments to functions. In several applications the usefulness of the proposed features are shown. In particular global properties of a model can be computed, such as total power, total mass, total center of mass, or kinetic and potential energy of a multi-body system. An important application is to bind behavioral models and requirement models in a convenient way, for example checking requirements for all instances of a class in a behavioral model, *without* changing the behavioral model.

Keywords: Array comprehension, array constructors, component iterators, binding, instance binding, class binding, total mass, total center of mass, total power.

1 Introduction

This article proposes two new Modelica language constructs to (a) access all instances of a given class and (b) to extract in a convenient way the desired information from such instances. The primary goal for these developments have been the enhancement of requirements modeling in Modelica, as proposed for example by (Jardin *et al.*, 2011; Bouskela *et al.*, 2015) and using it concretely in combination with the Modelica_Requirements library (Otter *et al.*, 2015). The difficulty here is to extract observations from a behavioral model (a) in a convenient way, (b) *without changing* the behavioral model, and (c) binding these observations to requirement models to assess the behavioral model.

Due to their generality, these new language constructs allow also other applications which cannot be expressed in a practical way with current Modelica. Most important, global properties, such as total center of mass of a mechanical system, or total power or energy of a system, can be calculated.

The language elements proposed in section 2 are supported in a Dymola prototype (Dassault Systèmes, 2015) and all the examples in this paper have been tested with it.

2 Proposals for new Language Elements

2.1 Component iterators

In section 10.4.1 of the Modelica Specification 3.3 (Modelica Association, 2014), array constructors with iterators are defined. For example,

```
Real v[:] = {i*i for i in 1:10};
```

generates a vector v with 10 elements and every element is the square of its index. Section 11.2.2.2 “Types as Iteration Ranges” states “The iteration range can be specified as `Boolean` or as an enumeration *type*”. It is proposed to generalize this scheme, so that a *class name* can be used as iterator expression and in every iteration the loop-variable is *one instance of this class*. The loop iterates over *all instances* of this class available in the simulation model, for example:

```
Real u[:] = {c.v for c in Class};
```

This construct can be used for example in the following way:

```
record Observation
  constant String name;
  parameter Real m;
  parameter Real v2[3];
  Real r2;
end Observation;

model Class
  parameter Real p=2;
  parameter Real v[3] = {-1,2.5,6};
  Real r;
  Real w[3];
  Boolean b;
  Integer i;
  ...
end Class;

model Submodel
  Class c1(p=3, v={1,-4,8});
  Class c2;
end Submodel;

model Model
  Submodel s1; Submodel s2;
  Integer i2[:] = {c.i+3 for c in Class};
  Observation obs[:] =
    {Observation(m=c.p, v2=c.v, r2=c.r,
                name=c.getInstanceName())
    for c in Class};
  Integer i3[:] = {c.i for c in ClassNotPresent};
end Model;
```

In every iteration of the `for` loops, the iterator variable `c` adopts the name of an instance of class `Class` present in `Model` (the complete model is inspected, independently where the iterator expression is present). The built-in operator `c.getInstanceName()` is expanded as the instance name of `c`. If no instance of a class is present in a model, such as for `ClassNotPresent`, then an array with zero dimensions is generated.¹

Therefore, the above model is equivalent to the following expanded form (showing that the extension can be formally defined by a rewriting rule):

```

model ModelExpanded
  Submodel s1;
  Submodel s2;

  Integer i2[:] = {s1.c1.i+3, s1.c2.i+3,
                  s2.c1.i+3, s2.c2.i+3};

  Observation obs[:] = {
    Observation(m=s1.c1.p, v2=s1.c1.v, r2=s1.c1.r,
               name="ModelExpanded.s1.c1"),
    Observation(m=s1.c2.p, v2=s1.c2.v, r2=s1.c2.r,
               name="ModelExpanded.s1.c2"),
    Observation(m=s2.c1.p, v2=s2.c1.v, r2=s2.c1.r,
               name="ModelExpanded.s2.c1"),
    Observation(m=s2.c2.p, v2=s2.c2.v, r2=s2.c2.r,
               name="ModelExpanded.s2.c2");

  Integer i3[0];
end ModelExpanded;
    
```

It is also proposed to extend the array constructor with guards to be able to restrict the set of instances using a built-in operator `instanceIn(..)`:

```

model ModelWithGuard
  Submodel s1;
  Submodel s2;
  Integer i2[:] = {c.i+3 for c in Class
                  if c.instanceIn(s1)};
end ModelWithGuard;
    
```

Here `c` takes the values “`s1.c1`” and “`s1.c2`”.

Naturally, there are restrictions of this new concept of component iterators, in particular:

- As class in the iterator only the specialized classes are possible that allow to construct *component* instances: `model`, `block`, `connector`, `record`, `operator record` (but not `package`, `function`, `operator function`).
- Component iterators can only be used in the specialized classes `model` and `block`.
- Component instances in `functions` are ignored (not returned) by component iterators.

2.2 Model instances as arguments to functions

It is proposed to generalize the calling mechanism of Modelica functions so that `model`, `block`, `connector`, `record` and `operator record` instances can be passed as arguments to functions, provided the instance is a subtype of the corresponding `record` function argument. Example:

```

model Submodel
  Real r1;
  Real r2;
  Integer i2
  Pin p1, p2;
protected
  Integer i1;
  ...
end Submodel;

record Record
  Real r1;
  Integer i2;
end Record;

function get
  input Record rec;
  output Real result;
algorithm
  result := rec.r1 + rec.i2;
end get;

model Model
  Submodel s1;
  Real r=get(s1);
end Model;
    
```

Note that input argument `rec` of function `get` expects an instance of record `Record` when calling the function. However, an instance of model `Submodel` is passed when calling this function. The semantics is that the function extracts the values of all elements of `s1` that are also present in record `rec`. The function call in the example is therefore equivalent to the Modelica 3.3 function call:

```

model ModelExpanded
  Submodel s1;
  Real r = get(Record(r1=s1.r1, i2=s1.i2));
end ModelExpanded;
    
```

Again, this language extension can be formally specified as rewriting rule. Since the rewriting is done locally, it seems like a minor convenience improvement. As the requirements binding applications in section 4 will show, this is not the case: The essential advantage is to define the elements that are extracted from a model only *once* (in the above example in the definition of function `get`) and the user of the function does not need to know which elements are extracted. If this function is used for many models, manually applying the rewriting would be no longer practical and would be error prone.

To summarize, the proposed language element is a short hand notation that is especially very convenient, if the record input argument to a function has many elements and the function is called many times for many model instances.

3 Application: Total Properties

In this section several applications are sketched how the language constructs from section 2 can be used in applications where total properties of a system model shall be computed.

¹ In order that it is possible to write generic code without knowing which classes are present in the simulation model, no error must be generated when a class is not present that is used as iterator.

3.1 Total mass

In a 3-dimensional mechanical system it is sometimes required to compute the total mass of a system. For example, to determine the complete mass of a vehicle, satellite, or robot from a behavioral model and compare it with the measured weight of the built system and/or with a CAD model. This allows to detect modeling errors, but it might also be needed to check a requirement (for example, the total aircraft weight must be at most xx kg).

When using the `Modelica.Mechanics.MultiBody` library, there are only two model classes where the mass of a body are defined:

- `MultiBody.Parts.Body`
- `MultiBody.Parts.PointMass`

All other specialized parts, like `Parts.BodyShape`, use an instance of `Parts.Body` and need therefore not to be handled specially. Model `TotalMass` computes the total mass of all bodies in a system.

```
model TotalMass "Compute total mass of system"
  import Modelica.Mechanics.MultiBody.Parts;
  import SI = Modelica.SIunits;
  final parameter SI.Mass m_total =
    sum({b.m for b in Parts.Body})
end TotalMass;
```

The assumption made here is that model `Parts.Body` has a parameter with name `m`. The sum of the `m` elements of all instances of `Parts.Body` is assigned to parameter `m_total`. This model can be, for example, used to compute the total mass of the `r3` robot from the Modelica Standard Library (see also Figure 1):

```
model TotalMassOfRobot "Compute total mass of r3 robot"
  import Modelica.Mechanics.MultiBody.Examples;
  extends TotalMass;
  extends Examples.Systems.RobotR3.fullRobot;
end TotalMassOfRobot;
```

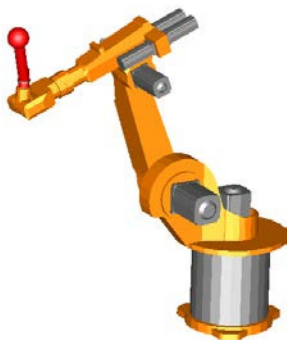


Figure 1. Animation of robot r3.

Simulating and inspecting the result file gives `m_total = 134.3 kg`

If the result is not as expected, it might be difficult to figure out the error in a larger system. It is then helpful to print out all the found masses, as performed in the next model:

```
model TotalMassWithLog Total mass with log"
  import Modelica.Utilities.Streams.print;
  import Modelica.Mechanics.MultiBody.Parts;
  import SI = Modelica.SIunits;

  final parameter SI.Mass m_total = sum(mObs[:].m);
protected
  record MassObservation
    String name "Name of body";
    SI.Mass m "Mass of body";
  end MassObservation;
  parameter MassObservation mObs[:] =
    {MassObservation(name=b.getInstanceName(),
                    m=b.m)
    for b in Parts.Body};

  equation
  when initial() then
    // print body names (mObs[:].name) and values
  end when;
end TotalMassWithLog;
```

Since the name of the body and its mass shall be extracted, a local record `MassObservation` is introduced and filled with the array comprehension language element. The built-in operator `getInstanceName()`, returns the names of the found body instances. When using the model for the `r3` robot, the following message is printed during initialization:

```
... Body masses:
mechanics.b0.body: 1 kg
mechanics.b1.body: 1 kg
mechanics.b2.body: 56.5 kg
mechanics.b3.body: 26.4 kg
mechanics.b4.body: 28.7 kg
mechanics.b5.body: 5.2 kg
mechanics.b6.body: 0.5 kg
mechanics.load.body: 15 kg
Total mass: 134.3 kg
```

3.2 Position vector to total center of mass

In space applications there is sometimes the need to determine the position of the total center of mass of a satellite, rocket, or space station. One reason is, for example, that a path planning software computed the desired trajectory (of the total center of mass) and the detailed mechanical model of the system shall start at a point on this trajectory. Another reason is when a robot is mounted on a free flying satellite system (as for example planned for repair operations). Then, movements of the robot do not change the position of the total center of mass, and a control system for grasping has to take this effect into account (and needs to know the total center of mass). With the definitions of Figure 2:

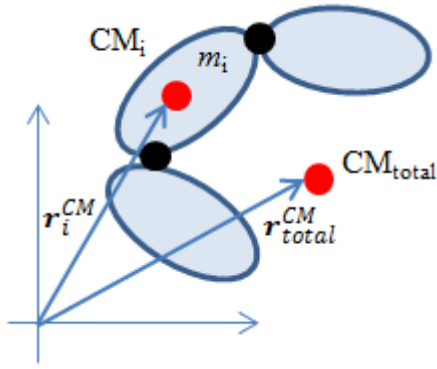


Figure 2. Computation of the total center of mass.

the well-known equation to compute the position of the total center of mass is (under the assumption that all absolute position vectors are resolved in the inertial frame):

$$\mathbf{r}_{total}^{CM} = \frac{\sum m_i \cdot \mathbf{r}_i^{CM}}{\sum m_i} \quad (1)$$

Model `TotalCenterOfMass` computes the absolute position of the total center of mass of all bodies in a system (`rCM_total`):

```

model TotalCenterOfMass
  import Modelica.Mechanics.MultiBody.Frames;
  import Modelica.Mechanics.MultiBody.Parts;
  import SI = Modelica.SIunits;

  SI.Mass m_total = sum(obs[:].m) "Total mass";
  SI.Position rCM_total[3] =
    {sum(m_rCM[:,j])/m_total for j in 1:3}
    "Total center of Mass";

protected
  record FrameObservation
    SI.Position r_0[3];
    Frames.Orientation R "Orientation matrix";
  end FrameObservation;

  record BodyObservation
    SI.Mass m "Mass of body";
    SI.Position r_CM[3] "Vector frame_a to CM";
    FrameObservation frame_a;
  end BodyObservation;

  function getObservations
    input BodyObservation obs;
    output BodyObservation result=obs;
    algorithm annotation(Inline=true);
  end getObservations;

  BodyObservation obs[:] =
    {getObservations(b) for b in Parts.Body};
  Real m_rCM[size(obs,1),3](unit="kg.m");
  equation
    for i in 1:size(obs,1) loop
      m_rCM[i,:] = obs[i].m*(obs[i].frame_a.r_0 +
        Frames.resolve1(obs[i].frame_a.R, obs[i].r_CM));
    end for;
  end TotalCenterOfMass;
    
```

The computation is performed in the following way:

1. The variables that shall be extracted from every body are defined in the protected section. These are the mass `m`, the local position vector `r_CM` from `frame_a` to the center of mass of the body, the absolute position vector `frame_a.r_0` from the inertial frame to `frame_a` and the orientation matrix `frame_a.R` transforming the inertial frame into `frame_a`. All these variables are defined in a record that has the same structure and uses the same names as used in model `Parts.Body`.
2. The central declaration of `obs` extracts the desired information from all instances of model `Parts.Body`:
`obs[:] = {getObservations(b) for b in Parts.Body};`
 There are two possibilities, either a record constructor is used to extract the body variables (as in `TotalMassWithLog`), or a function is used as above (`getObservations(b)`) and one `Parts.Body` instance is passed to the function. With the new semantics of section 2.2, the tool extracts all variables from the instances and copies them into instances of the `BodyObservation` record.
3. Once the variables from all instances of `Parts.Body` are extracted, it is rather straightforward to compute the desired position vector to the total center of mass. This requires to transform all body-fixed position vectors `obs[i].r_CM` into the inertial frame, add the absolute position vectors at the body frames `obs[i].frame_a.r_0`, and use equation (1).

This model can be, for example, used to compute the position vector to the total center of mass of the `r3` robot, see Figure 1:

```

model TotalCenterOfMassOfRobot
  "Compute total center of mass of r3 robot"
  import Modelica.Mechanics.MultiBody.Examples;
  extends TotalCenterOfMass;
  extends Examples.Systems.RobotR3.fullRobot;
  end TotalCenterOfMassOfRobot;
    
```

A simulation produces the result in Figure 3.

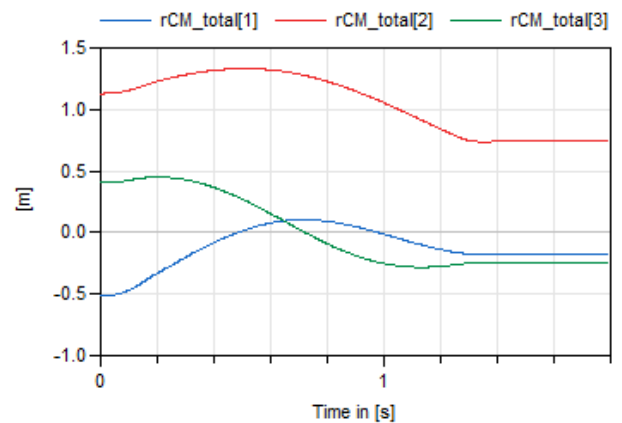


Figure 3. Simulation results to compute the position vector to the total center of mass of the `r3` robot.

4 Application: Requirements Binding

In this section a class of applications is discussed how to bind requirement models in a convenient way to behavioral models using the language constructs from section 2.

4.1 Overview

In (Jardin et al., 2011) a concept was developed to model properties and requirements in Modelica. This was significantly enhanced in (Bouskela et al., 2015) and a sophisticated Modelica library for this approach was developed in (Otter et al., 2015).

In industry, requirements are usually defined in natural language, such as²

- *When in operation, pumps shall not cavitate*
(= the pressure in a pump must be larger than a minimum pressure)
- *In flight, with only one engine running, the air distribution circuit shall provide nominal performance.*
- *After three failures of starting an engine, the APU (Auxiliary Power Unit) must be started.*

The basic idea is to provide a suitable Modelica library to model such requirements in a formal way with Modelica, see (Otter et al., 2015) for details. There are the following key requirements from industry (Bouskela et al., 2015):

1. The requirement models are developed independently from the behavioral models that shall be checked. The reason is (a) that requirements shall be formally specified before designing the system (and therefore a behavioral model is not yet available), and (b) that requirements are defined from system architects which are not the simulation specialists building up the behavioral models. As a consequence, the variables used in requirement models need not be the same (not even the data type) as the (corresponding) variables in the behavioral model.
2. When associating requirement models to behavioral models (in order to check the behavioral models), it is usually not possible or not allowed to change or adapt the code of the behavioral models.

Based on these restrictions there is a fundamental issue how to extract variables from a behavioral model (these variables are called “observation” variables below) and assign them as inputs to the requirement models. This process is called “Binding” in the sequel. In (Jardin et al., 2011) Modelica buses have been used for the “Binding”. This violates the restrictions above since the behavioral model must be modified and the

variable names in the behavioral and requirement models must be identical. Furthermore, in larger use cases of EDF and Dassault Aviation it turned out that this is not a practical approach because it is also much too inconvenient to use.

There have been also other proposals how to define the “Binding”, such as (Schamai, 2013). Still, until now, no satisfactory approach is known to be used conveniently in Modelica. In the rest of this section it is shown that the proposed new language elements of section 2 provide a convenient and powerful way to define the “Binding”.

4.2 Instance binding

The goal is to check the following requirement for all pumps present in a system:

When in operation, a pump shall not cavitate.

This requirement can be checked with the following model³:

```
record PumpObservation
  constant String name "Name of pump";
  Boolean inOperation "= true, if in operation";
  Boolean cavitate "= true, if pump cavitates";
end PumpObservation;

model PumpRequirements
  import Modelica.Utilities.Streams.print;
  input PumpObservation obs[:];
equation
  for i in 1:size(obs,1) loop
    when obs[i].inOperation and obs[i].cavitate then
      print("... warning: pump " + obs[i].name +
        " is cavitating during operation");
    end when;
  end for;
end PumpRequirements;
```

The requirement definition above is independently of the construction of the pump and how the values of the Boolean variables `inOperation` and `cavitate` are determined from the behavioral model. For a concrete pump, here from:

Modelica.Fluid.Machines.PrescribedPump

a function is used to map observation variables of an instance of `PrescribedPump` to the variables needed by the requirement model:

```
function fromPrescribedPump
  input PrescribedPumpObservation obs;
  input String name;
  input Modelica.SIunits.Pressure p_cavitate=0.99e5;
  output PumpObservation result(
    name = name,
    inOperation = obs.N_in > 0.1,
    cavitate = obs.port_a.p <= p_cavitate or
      obs.port_b.p <= p_cavitate);
```

² These and further examples from this section are from (Bouskela et al., 2015) or (Otter et al., 2015)

³ In case of violation, only a warning message is printed. In (Otter et al., 2015) a more involved handling is performed.

```

protected
record PortObservation
  Modelica.SIunits.Pressure p;
end PortObservation;
record PrescribedPumpObservation
  Real N_in(unit="1/min");
  PortObservation port_a;
  PortObservation port_b;
end PrescribedPumpObservation;
algorithm
  annotation(GenerateEvents=true);
end fromPrescribedPump;
    
```

Here **PrescribedPumpObservation** is a record declared internally in the function that defines which variables shall be extracted from an instance of the PrescribedPump model. In this case, these are N_{in} , the speed of the pump shaft, as well as $port_a.p$ and $port_b.p$, the pressures at the pump ports. This record is used as input argument together with the name of the pump. As output argument, an instance of the **PumpObservation** record is used and via a record constructor the variables from the **PrescribedPumpObservation** are mapped to the **PumpObservation** output argument.

When using the record constructor, relations are present, such as $obs.N_{in} > 0.1$. With a normal function, this would lead to an error during translation, because (a) relations in functions do not generate events, (b) this function is called in the continuous-time part of Modelica and (c) in Modelica it is not allowed that Boolean variables can change during continuous-time integration. This problem is resolved with the annotation `GenerateEvents=true`. This is a standard Modelica annotation and defines that relations in this function generate events. The effect is that the Boolean variables can only change at event points.

The **PumpRequirements** model and the mapping function from a **PrescribedPump** to this model is now evaluated with example **BatchPlant_StandardWater** form the Modelica Standard Library. A screen shot of this model is shown in Figure 4. This model has two instances of model **PrescribedPump**, named P1 and P2 (at the bottom of the diagram). This system is checked with the following model:

```

model CheckPumpsOfBatchPlant
  import Modelica.Fluid.Examples.AST_BatchPlant;
  extends AST_BatchPlant.BatchPlant_StandardWater;

  PumpRequirements req(obs=
    {fromPrescribedPump(P1,"P1"),
     fromPrescribedPump(P2,"P2")});
end CheckPumpsOfBatchPlant;
    
```

As can be seen, an instance of the **PumpRequirements** model is defined. The pump instances P1 and P2 from the **BatchPlant_StandardWater** model are passed as arguments to function `fromPrescribedPump`. With the proposed language feature of 2.2, observation variables

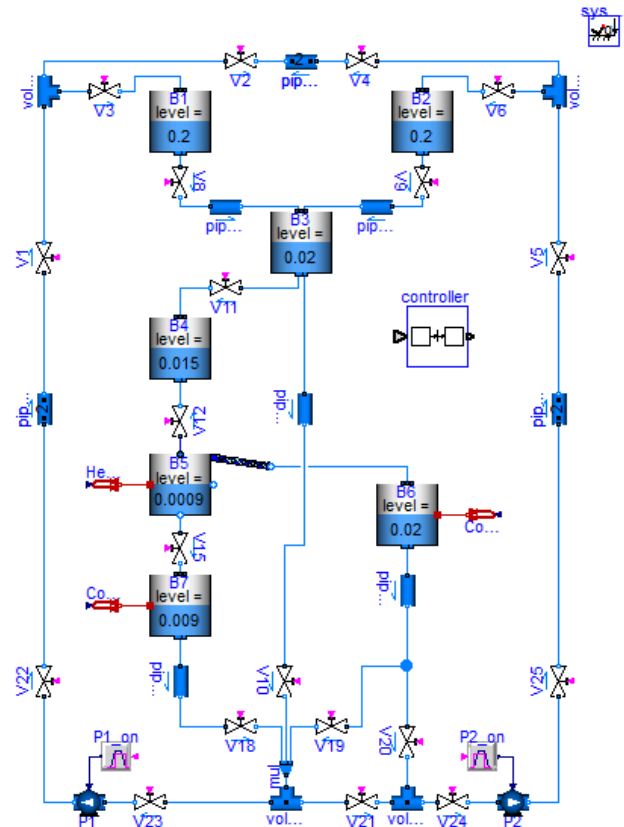


Figure 4. Example model **BatchPlant_StandardWater** form the Modelica Standard Library.

are extracted from the pumps and are transformed as needed from the requirement models.

Note, as required the behavioral model (= **BatchPlant_StandardWater**) is not modified and the observation variables used in the behavioral model and the requirement model might be different. Simulation results are shown in Figure 5.

As can be seen, one of the pumps is cavitating once. As a result, the log window contains a warning message:

... warning: pump P1 is cavitating during operation

There is always the need to specify for one or more individual instances specific requirements (for example, “at least one pump present in room A must always be in operation”⁴), and then the approach above, also called instance binding, has to be applied.

However, there are also requirements that hold for many instances, and the instance binding may then become inconvenient. In the next section this case is handled by “class binding”.

⁴ In this case a vector of pumps must be passed to the requirement model consisting of the pump instances present in room A.

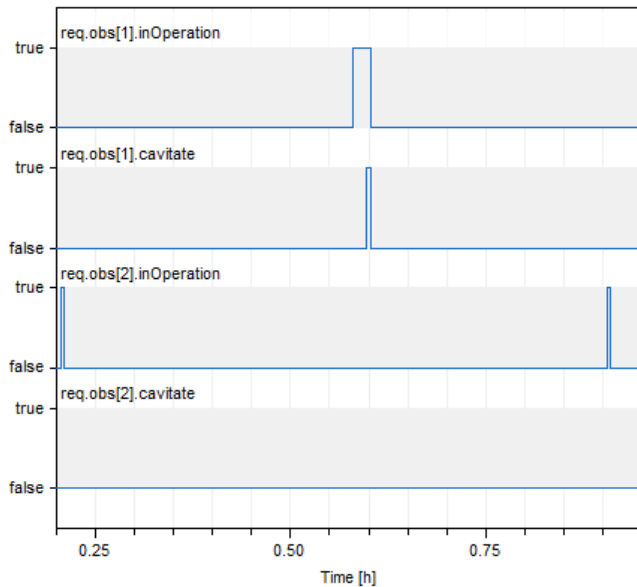


Figure 5. Simulation results for the requirements model of `BatchPlant_StandardWater` of Figure 4.

4.3 Class binding

In case a requirement holds for all instances of a class, the array of observations need not be defined manually but can be generated with the array comprehension for classes of section 2.1. The previous example can then be defined as:

```

model CheckPumpsOfBatchPlantWithForLoop
  import Modelica.Fluid.Examples.AST_BatchPlant;
  extends AST_BatchPlant.BatchPlant_StandardWater;

  PumpRequirements req(obs=
    {fromPrescribedPump(p, p.getInstanceName())
      for p in Modelica.Fluid.Machines.PrescribedPump});
end CheckPumpsOfBatchPlantWithForLoop;

```

Note, that this model generates requirement checks for **any number of pumps** in the circuit. With the planned guard on for-loops, it would also be possible to limit the for-loop to instances of the desired class in a specific sub-model.

4.4 Advanced class binding

Class-binding becomes more involved if instances for two or more classes have to be treated simultaneously. Here is a sketch of two different approaches based on the scenario defined in (*Bouskela et al, 2015*):

A pump might be built up from several components, for example with a centrifugal pump and with an electric motor that drives the centrifugal pump. However, the requirements from section 4.2, `PumpRequirements`, are always the same, independently of the underlying technology of the pump.

Assume that a cooling circuit is defined by two subsystems that contain each three pumps built up by centrifugal pump and electric motor components:

```

model Subsystem
  CentrifugalPump P1;
  ElectricMotor M1;

  CentrifugalPump P2;
  ElectricMotor M2;

  CentrifugalPump P3;
  ElectricMotor M3;
  ....
end Subsystem;

model CoolingSystem
  Subsystem subsystem1;
  Subsystem subsystem2;
end CoolingSystem;

```

The goal is to check the pumps. In order to do this one has to collect observation variables, say, from P1 and M1 and pass them to `PumpRequirements`. This is straightforward for instance binding, but more complicated if code for any number of instances shall be implemented.

The essential difficulty is that information is missing: It is not known from the `Subsystem` definition whether P1 and M1 or P1 and M2 or P1 and M3 form the pump. It might be possible to deduce this information from the connection of the components but it seems quite complicated to provide language elements to the user such that he/she can implement code to deduce the connection structure. Furthermore, even then there might be not a unique solution because the motor M1 might not be directly connected to P1 (but via another auxiliary component), or two motors might be connected to P1, but only one of them is relevant for the requirement model.

The solution proposed here is to add more information. If it is not allowed or not possible to modify the behavioral model, the only way is to list the instances that belong together. This is performed in the following model:

```

model CheckCoolingSystem
  extends CoolingSystem;
  constant String pumpMotorAssociations[:,3]=
    ["subsystem1", "P1", "M1";
     "subsystem1", "P2", "M2";
     "subsystem1", "P3", "M3";
     "subsystem2", "P1", "M1";
     "subsystem2", "P2", "M2";
     "subsystem2", "P3", "M3"];

  constant Integer motorIndices[:]:=
    associateCPumpsAndEMotorsByNames(
      {p.getInstanceName() for p in CentrifugalPump},
      {m.getInstanceName() for m in ElectricMotor},
      pumpMotorAssociations, getInstanceName());

  PumpRequirements req(obs=
    fromCPumpAndEMotor(
      {fromCPump(p) for p in CentrifugalPump},
      {fromEMotor(m) for m in ElectricMotor},
      motorIndices));
end CheckCoolingSystem;

```

Array `pumpMotorAssociations` has three columns: The first column contains the path name of the subsystem in which the pump is present, such as "subsystem2". The second and third columns contain the names of the centrifugal pump and the electric motor that form the pump, such as "P3", "M3". This array has to be manually constructed for the circuit at hand.

With function `associateCPumpsAndEMotorsByNames` the association of centrifugal and electric motor instances is determined once during translation of the model and the result is assigned to the constant Integer array `motorIndices`, such that if centrifugal pump `i` is associated with electric motor `j`, then `motorIndices[i]=j`.

In order to map the observations from the behavioral model to the `PumpRequirements` model several new mapping functions are needed. For example, function `fromCPumpAndEMotor` can be implemented as:

```
function fromCPumpAndEMotor
  input PumpObservation_cavitate pObs[:];
  input PumpObservation_inOperation mObs[:];
  input Integer motorIndices[size(pObs,1)];
  output PumpObservation obs[size(pObs,1)];
algorithm
  for i in 1:size(pObs,1) loop
    obs[i].cavitate := pObs[i].cavitate;
    obs[i].inOperation :=
      mObs[motorIndices[i]].inOperation;
  end for;
end fromCPumpAndEMotor;
```

As can be seen, the `motorIndices` vector is used to extract observation variables from the electric motor observations `mObs[motorIndices[i]]` that are associated with the corresponding centrifugal pump observations `pObs[i]`.

In case it is possible to modify the behavioral model to be checked (here: `CoolingSystem`), another approach might be more convenient and less error prone: Every component gets an additional unique Integer identification number, called "id". A centrifugal pump and an electric motor belong together and form one pump, if both have the same "id". It is not allowed that any other pump in the circuit has the same "id". The circuit can then be modelled in the following way:

```
model SubsystemWithID
  CentrifugalPumpWithID P1 (id=1);
  ElectricMotorWithID M1(id=1);
  CentrifugalPumpWithID P2 (id=2);
  ElectricMotorWithID M2(id=2);
  CentrifugalPumpWithID P3 (id=3);
  ElectricMotorWithID M3(id=3);
end SubsystemWithID;

model CoolingSystemWithID
  SubsystemWithID subsystem1;
  SubsystemWithID subsystem2(P1(id=4),M1(id=4),
    P2(id=5),M2(id=5),
    P3(id=6),M3(id=6));
end CoolingSystemWithID;
```

The checking of the requirements can be performed as:

```
model CheckCoolingSystemWithID
  extends CoolingSystemWithID;

  constant Integer motorIndices[:]=
    associateCPumpsAndEMotorsByID(
      {p.id for p in CentrifugalPumpWithID},
      {m.id for m in ElectricMotorWithID});

  PumpRequirements req(obs=
    fromCPumpAndEMotor(
      {fromCPump(p) for p in CentrifugalPumpWithID},
      {fromEMotor(m) for m in ElectricMotorWithID},
      motorIndices));
end CheckCoolingSystemWithID;
```

Since the information about the association of centrifugal pump and electric motor is within the behavioral model, the code for the requirement check in `CheckCoolingSystemWithID` is generic. Function `associateCPumpsAndEMotorsByID` determines the same index vector `motorIndices` as before. The implementation of this function is however simpler:

```
function associateCPumpsAndEMotorsByID
  input Integer pumpIds[:];
  input Integer motorIds[:];
  output Integer motorIndices[size(pumpIds,1)];
algorithm
  for i in 1:size(pumpIds,1) loop
    for j in 1:size(motorIds,1) loop
      if motorIds[j] == pumpIds[i] then
        motorIndices[i] :=j; break;
      elseif j == size(motorIds,1) then
        assert(false, "id's are wrong");
      end if;
    end for;
  end for;
end associateCPumpsAndEMotorsByID;
```

In order to provide better diagnostics in case of an error, it is useful to pass the instance names of the centrifugal pumps and of the electric motors also to this function. For simplicity this was not done above. Furthermore, it should also be checked, that the id's are unique.

5 Summary

This paper proposes two new Modelica language elements to extract information from a model in a convenient way. This opens up new applications of Modelica that could not be practically handled before. The language elements and the sketched applications have been evaluated and tested with a Dymola prototype.

Acknowledgements

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

Helpful discussions with Daniel Bouskela, Nguyen Thuy, Audrey Jardin (EDF), Eric Thomas, Maxim Payelleville (Dassault Aviation), Wladimir Schamai (Airbus Defence and Space), Peter Fritzson, Lena Buffoni (PELAB), Alfredo Garro and Andrea Tundis (UNICAL) on the “Requirements Binding” application of section 4 are appreciated.

References

- Bouskela D., Thuy N., Jardin A. (2015): **D2.1.1 – Modelica extensions for properties modelling, Part II: Modeling Architecture for the Design Verification against System Requirements**. Internal report, ITEA2 MODRIO project, March 2015.
- Dassault Systèmes (2015): **Dymola 2016**.
<http://www.Dymola.com>
- Jardin A., Bouskela D., Thuy N., Ruel N., Thomas E., Chastanet L., Schoenig R., Loembé S. (2011): **Modelling of System Properties in a Modelica Framework**. Proceedings 8th Modelica Conference, Dresden, Germany, March 20-22., pp. 579-592. Download: <http://www.ep.liu.se/ecp/063/065/ecp11063065.pdf>
- Modelica Association (2014): **Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.3, Revision 1**, June 11, 2014. Download: <https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf>
- Otter M., Thuy N., Bouskela D., Buffoni L., Elmqvist H., Fritzson P., Garro A., Jardin A., Olsson H., Payelleville M., Schamai W., Thomas E., Tundis A. (2015): **Formal Modeling and Automatic Verification of Requirements**. Proceedings 11th Modelica Conference, Versailles, France, Sept. 21-23.
- Schamai, W. (2013): **Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool**. Ph.D. Thesis, No. 1547, University of Linköping. Download: <http://liu.diva-portal.org/smash/record.jsf?pid=diva2:654890>

Flattening of Modelica State Machines: A Practical Symbolic Representation

Bernhard Thiele¹ Adrian Pop¹ Peter Fritzson¹

¹PELAB, Linköping University, Sweden, {bernhard.thiele, adrian.pop, peter.fritzson}@liu.se

Abstract

Modelica 3.3 introduced dedicated built-in language support for *state machines* that was inspired by semantics known from *Statechart* and *mode automata* formalisms. The specification describes the semantics of these constructs in terms of data-flow equations that allows it to be related to the Modelica DAE representation which is the conceptual intermediate format of Modelica code after instance creation (flattening). However, a complete transformation of state machine constructs into data-flow equations at the stage of flattening requires an early commitment to implementation details that potentially hinders model optimizations at subsequent translation phases. Also, due to the required substantial model transformation the semantic distance between the original source model and the *flattened* representation is rather large. Hence, this paper proposes a more versatile symbolic representation for flattened state machine constructs that preserves the state machine's composition structure and allows postponing optimizations to subsequent compiler phases.

Keywords: state machine, mode automata, flattening, compilation

1 Introduction

The scope of the Modelica specification is briefly stated in (Modelica Association, 2012, Section 1.2):

The semantics of the Modelica language is specified by means of a set of rules for translating any class described in the Modelica language to a flat Modelica structure. A class must have additional properties in order that its flat Modelica structure can be further transformed into a set of differential, algebraic and discrete equations (= hybrid DAE). Such classes are called simulation models.

A typical compilation process for a Modelica language tool is structured as depicted in Figure 1. *Flat Modelica* is an intermediate representation which is further elaborated into a representation from which optimized simulation code can be generated. Conceptually, flat Modelica

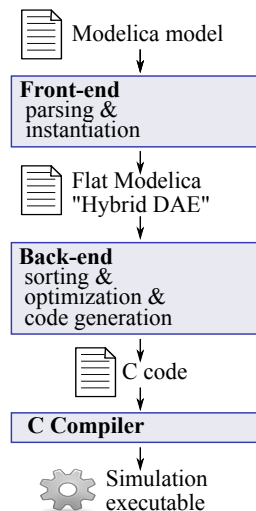


Figure 1. Outline of a typical compilation process for a Modelica language tool.

is closely related to a *hybrid DAE* (hybrid Differential Algebraic Equation) representation. This relationship is discussed in (Modelica Association, 2012, Appendix C). The mapping from flat Modelica to a hybrid DAE is a powerful concept, since it provides a mathematical foundation for the semantics of flat Modelica.

Modelica 3.3 introduced dedicated built-in language support for *clocked state machines* that was inspired by semantics known from *Statechart* (Harel, 1987) and *mode automata* formalisms (Maraninchi and Rémond, 2003), particularly the mode automata variant implemented in the Lucid Synchrone 3.0 language (Pouzet, 2006).

The Modelica specification describes the semantics of state machines by a set of rules that allows relating state machines to purely data-flow based Modelica code (Modelica Association, 2012, Chapter 17)¹. Hence, state machine constructs are reduced to data-flow equation constructs for which the flattening process is already described in other parts of the language specification.

From this perspective it is natural to perform a complete transformation of state machine constructs to data-

¹A more accessible presentation of Modelica state machines with additional examples can be found in (Fritzson, 2014, Chapter 13).

flow equations during the flattening process, so that the resulting flat Modelica can be directly related to a flat hybrid DAE. However, a complete transformation of state machine constructs into data-flow equations at the stage of flattening requires an early commitment to implementation details. This commitment makes model optimizations more difficult at subsequent translation phases. Additionally, the required substantial model transformation renders the semantic distance between the source code and the flattened representation rather large which reduces the value of flat Modelica as a traceable human checkable intermediate model representation.

2 State Machine Flattening in Current Tools

At the time of writing, only Dymola² provides full support for Modelica state machines. A presentation about an early (incomplete) prototype implementation for OpenModelica³ was given in the OpenModelica Annual Workshop (Thiele, 2015). The flat Modelica code resulting from State Machines in Dymola resembles the code generated by the above-mentioned OpenModelica prototype.

The simple state machine example presented in the original Modelica state machine paper by Elmqvist et al. (2012) is reused for illustrating the relation between the state machine Modelica code and the generated flat Modelica representation. Figure 2 shows the graphical representation of that state machine as well as a plot of its variable i for 30 seconds of simulation. The state machine

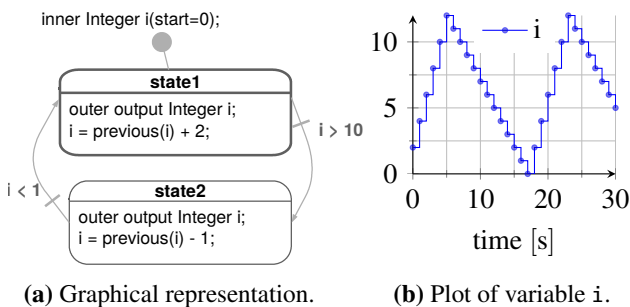


Figure 2. Simple state machine.

extension is based on Modelica’s synchronous language elements extension. The discrete-time equations within a state machine are based on *clocked* variables and all variables and equations within it must be associated with the same clock. The semantics are:

- Equations are active at *clock* ticks generated by the clocks associated with the equations. If no clock is associated a default clock is used. In the example of Figure 2 the default clock used is a periodic clock with 1.0s sampling period.

²<http://www.dymola.com/>

³<https://www.openmodelica.org/>

- The variable i is a *shared* variable between the two states `state1` and `state2`⁴.
- The example uses “*delayed*” transitions⁵, hence the transitions do not fire immediately if the associated condition on i evaluates to **true**. Instead they fire at the subsequent clock tick.
- Furthermore, the transitions are declared as “*reset*” transitions⁶. Reset transitions reinitialize the “states” of their target states, *i.e.*, set the values of the state variables “owned” by those states to their start values and reset nested state machines. Within the considered example `state1` and `state2` declare access to the *outer* state variable i . Outer variables are *not* reset if entering the state. Hence, for the considered example it makes no difference whether or not a transitions is a “reset” transition.

The Modelica model (ignoring annotations) that corresponds to the graphical representation of Figure 2 is displayed in Listing 1.

Listing 1. Modelica model corresponding to Figure 2.

```

model SimpleSM "Simple state machine"
  inner Integer i(start=0);
  block State1
    outer output Integer i;
    equation
      i = previous(i) + 2;
    end State1;
  State1 state1;
  block State2
    outer output Integer i;
    equation
      i = previous(i) - 1;
    end State2;
  State2 state2;
  equation
    transition(state1,state2,i > 10,
      immediate=false,reset=true,
      synchronize=false,priority=1);
    transition(state2,state1,i < 1,
      immediate=false,reset=true,
      synchronize=false,priority=1);
  initialState(state1);
end SimpleSM;
    
```

The flat Modelica representation generated by Dymola 2015 FD01 is reproduced in a slightly reformatted form (to save space) in Listing 2. Note that there is a discrepancy between the number of variables (three) and the

⁴The “outer” prefix declares that an element instance with the same name, but using prefix “inner” within the enclosing instance hierarchy is referenced.

⁵Delayed transitions are depicted by a perpendicular line close to the “from”-state. For immediate transitions this line is close to the “to”-state.

⁶Reset transitions are depicted by a filled arrow head (otherwise an open arrow head is used).

Listing 2. Flat Modelica model generated from the simple state machine model defined in Listing 1.

```

model SimpleSM
Integer i(start = 0);
Integer state1.i = i;
Integer state2.i = i;

// Equations and algorithms

// Component state1
// class SimpleSM.State1
equation
  state1.i = previous(state1.i)+2;

// Component state2
// class SimpleSM.State2
equation
  state2.i = previous(state2.i)-1;

// Component
// class SimpleSM
equation
  transition(state1,state2,i > 10,
    false,true,false,1);
  transition(state2,state1,i < 1,
    false,true,false,1);
  initialState(state1);
end SimpleSM;

```

number of equations (four). This imbalance is solved by the state machine semantics that require that outer output variables of each state are solved for and that for each such variable a single definition is formed. Hence, after substituting the alias variables in the example and merging outer variables this can be reduced to one variable and one equation, e.g.,

```

i := if activeState(state1) then
  previous(i)+2
elseif activeState(state2) then
  previous(i)-1 else previous(i)

```

The last else branch can never be reached in this particular example, but it illustrates that a state variable will simply keep its current value if there is no state active in which an equation for that variable is defined.

Deducing the equation transformation above from the flat Modelica representation is an essential step for relating flat Modelica to a valid DAE representation. Arguably, the information about this necessary equation transformation is present in the flat Modelica in a highly implicit fashion which is not only elusive for human perception, but also difficult to reason about mechanically.

One can deduce that `state1` and `state2` are states and that `state1.i` and `state2.i` are variables declared in the respective states. However, in the flat representation it is not obvious that they are shared variables and that two of their defining equations need to be merged into a *single* definition to form a valid system of equations (otherwise there is one equation too many).

As an example of this ambiguity in the flat representation consider the *invalid* model from Listing 3 that actually has one equation too many, but still has (apart from some comments) the same flattened representation as the simple state machine model from Listing 1 (compare the respective flat Modelica representations in Listing 4 and Listing 2). Trying to simulate the model from Listing 3

Listing 3. Invalid Modelica code that has a similar flat representation as the (valid) code from Listing 1.

```

model InvalidSM "Invalid model, but
  instructive flat representation"
  inner Integer i(start = 0);
  block State1
    input Integer i; // no shared variable!
  end State1;
  State1 state1(i=i);
  block State2
    input Integer i; // no shared variable!
  end State2;
  State2 state2(i=i);
equation
  // one equation too many
  state1.i = previous(i) + 2;
  state2.i = previous(i) - 1;
  transition(state1,state2,i > 10,
    immediate=false,reset=true,
    synchronize=false,priority=1);
  transition(state2,state1, i < 1,
    immediate=false,reset=true,
    synchronize=false,priority=1);
  initialState(state1);
end InvalidSM;

```

in Dymola fails with a (correct) error message complaining about more Integer equations than Integer variables (Dymola still generates the flat Modelica representation for the model since the model can be instantiated, but it cannot be translated due to the overconstrained equation system).

The important point is that solely by inspecting the flat Modelica representation that Dymola generates it is not obvious whether it corresponds to a valid or an invalid model: the flat Modelica representation in Listing 4 is, apart from additional comments, similar to the flat Modelica representation in Listing 2.

This example should illustrate that it is quite intricate to give the correct semantics of flat Modelica state machine representations generated by current Modelica tools. The example used the flat Modelica representation generated by Dymola 2015 FD01, but similar reasoning applies to the flat Modelica generated by the first prototypical support for state machines implemented in OpenModelica. A deliberately simple example was used in order to keep the discussion comprehensible.

To give the correct semantics of state machines encoded in the considered flat Modelica representation, it is necessary to deduce structural information regarding the state machine composition, e.g.,

Listing 4. Flat Modelica model generated from the (invalid) model defined in Listing 3.

```

model InvalidSM
Integer i(start = 0);
Integer state1.i = i;
Integer state2.i = i;

// Equations and algorithms

// Component state1
// class InvalidSM.State1
// extends InvalidSM
equation
  state1.i = previous(i)+2;
// end of extends

// Component state2
// class InvalidSM.State2
// extends InvalidSM
equation
  state2.i = previous(i)-1;
// end of extends

// Component
// class InvalidSM
equation
  transition(state1,state2,i > 10,
    false,true,false,1);
  transition(state2,state1,i < 1,
    false,true,false,1);
  initialState(state1);
end InvalidSM;

```

- associate assignment equations for state variables to corresponding states,
- recover the hierarchical state machine structure,
- identify shared variables,
- identify in which equations shared variables are used in an assignment context, and finally,
- deduce which assignment equations need to be merged.

However, experience from the first prototypical implementation in OpenModelica suggest that it is hard to automatically reconstruct this information. from the flattened representation without propagating further structural information about the model from the front-end to the back-end. This crucial additional information is not visible in the flat Modelica representation. The following sections will therefore discuss a symbolic representation for flattened state machine constructs that makes such structural information explicitly available within the flat Modelica model.

3 Practical Symbolic Representation

Different approaches have been experimented with in order to find an adequate symbolic representation. One important requirement is that the representation should be flexible enough for future incorporation of continuous-time equations. Hence, it should be general enough to allow for multi-mode DAE/ODE modeling resembling the style that was advocated by Elmqvist et al. (2014) and Bouissou et al. (2014). In a first approach it was investigated whether symbolic representations developed in the context of *hybrid automata* modeling and verification (Alur et al., 1993) could be adapted and reused in a Modelica context.

The basic idea in this first approach was to generate flat state machine representations and use *interconnection relations* to describe parallel and hierarchical compositions. This idea was motivated by a versatile notion of composition described by Tabuada (2009) in the context of hybrid system modeling. However, the representations became large (depending on the example about twofold the size compared to the representation proposed below), appeared rather artificial in the context of Modelica, and required many decisions during the flattening process that seem to be better postponed to the back-end.

Therefore, a more lightweight approach is proposed. It is based on the following basic ideas:

- Preserve the state machine hierarchy by introducing the notions of **stateMachine** and **state**.
 - stateMachine** Consists of a set of mutually exclusive states that are related by transitions (flat state machine).
 - state** Consists of variable declarations and equations associated to that state. May have nested state machines.
- Generate the equations necessary for merging shared variables from mutual exclusive states.

The resulting representation has the property that the number of equations and variables must be equal for a valid system.

Instead of the terms “stateMachine” and “state” one might prefer to use “automaton” and “mode” which capture that a system operates in a certain *mode* and that the automaton logic allows to change the active mode. However, the proposed terms correspond to the terms that are used in the state machine chapter of the Modelica 3.3 specification.

3.1 Simple State Machine Example

With the proposed extension, the flat representation of the simple state machine example from Listing 1 translates to the flat Modelica displayed in Listing 5. Note that two auxiliary variables \$state1.i, \$state2.i have been

Listing 5. Extended flat Modelica model proposed to be generated for the simple state machine model defined in Listing 1.

```

class SimpleSM
Integer i(start=0);
stateMachine smOf.state1
state state1
Integer $state1.i;
equation
$state1.i = previous(i) + 2;
end state1;

state state2
Integer $state2.i;
equation
$state2.i = previous(i) - 1;
end state2;
end smOf.state1;

equation
i = if activeState(state1) then $state1.i
elseif activeState(state2) then $state2.i
else previous(i);
transition(state1,state2,i > 10,
false,true,false,1);
transition(state2,state1,i < 1,
false,true,false,1);
initialState(state1);
end SimpleSM;
    
```

introduced. They are substitutes for `state1.i`, `state2.i` in the respective states and are used in the generated variable merging equation. However, in case `state1.i` and `state2.i` appear as argument in a `previous(..)` operator the variable `i` is substituted instead.

The “\$” prefix shall denote an auxiliary variable that is related to the subsequent variable name, but not strictly identical (e.g., `i = state1.i = state2.i`, but `i ≠ $state1.i ≠ $state2.i`).

Compared to the flat Modelica representation of Listing 2, the semantics of the simple state machine example is more explicitly represented in the flat Modelica of Listing 5 which re-enables the possibility to interpret the flat Modelica representation in a meaningful manner. Note that Listing 5 has the desirable property that the number of equations equals the number of unknowns since the variable merging equation (i.e., the equation for `i`) is made explicitly visible.

3.2 Hierarchical State Machine Example

The hierarchical state machine example shown in Figure 3 is motivated by the example given by Maraninchi and Rémond (2003). The state machine receives a stream of input values `i` and `j` and computes the variables `x`, `y`, and `z`. For this example the input values have been set to the constant values `i=true` and `j=false`.

Listing 6 shows how information about the structural composition for the hierarchical state machine from Fig-

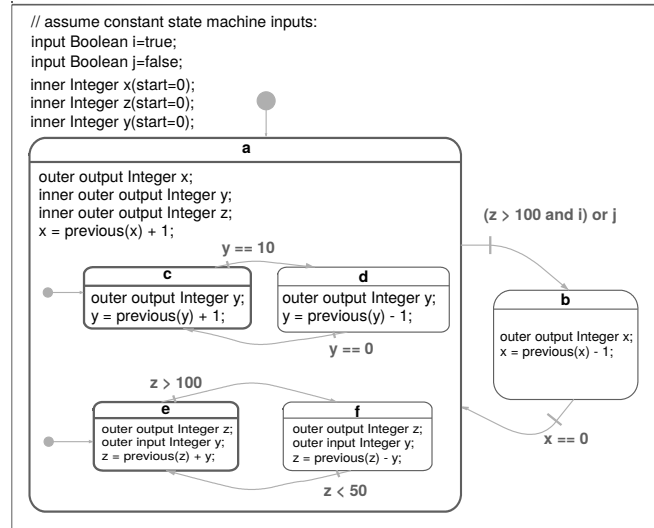


Figure 3. Hierarchical and parallel composition example motivated from Maraninchi and Rémond (2003).

ure 3 is preserved in the proposed flat Modelica representation. The complete listing for the flat Modelica representation is given in Appendix A.

3.3 Summary of Rules

The rules for mapping from state machines specified in Modelica to the proposed flat representation can be summarized as follows:

- Any class instance `x` that appears as argument in an `initialState(..)` or `transition(..)` operator results in a section `state x ... end x`; in the flat Modelica representation.
- Any class instance `x` that appears as argument in an `initialState(..)` operator results in a section `stateMachine smOf.x ... end smOf.x`; in the flat Modelica representation⁷.
- States that are connected by transition relations are collected in a `stateMachine` section. The identifier of the `stateMachine` section encodes the component reference of the initial state of that state machine. Hence, a `stateMachine` section collects states that belong to the same *flat* state machine.
- For any **outer output** or **inner outer output** variable declaration `x`, an auxiliary variable `$x` is introduced and all occurrences of `x` are replaced by `$x` unless `x` appears as argument in a `previous(..)` operator in which case `x` is replaced by its corresponding most **inner** component reference⁸.

⁷The name following the “stateMachine” construct has no significant semantics and could be also omitted, e.g., “stateMachine ... end;”.

⁸Hence, it is replaced by the most inner component reference and not by a references to an intermediate “inner outer output” declaration.

Listing 6. Preservation of state machine composition information for the hierarchical state machine from Figure 3.

```

class HierarchicalSM
  stateMachine smOf.a
    state a
      stateMachine smOf.a.c
        state a.c
          ...
        end a.c;
      state a.d
        ...
      end a.d;
    end smOf.a.c;
  stateMachine smOf.a.e
    state a.e
      ...
    end a.e;
  state a.f
    ...
  end a.f;
end smOf.a.e;
...
end a;
state b
...
end b;
end smOf.a;
...
end HierarchicalSM;
    
```

- **outer** variables that are not declared as **output** are replaced by their corresponding **inner** component references.

- Equations for merging shared variables are introduced according to the following rules:

- For any **inner** (or **inner outer**) variable that is referenced by an **outer** (or **inner outer**) variable declaration in one or more states (within the same hierarchy) of a **stateMachine** section, a merging equation is formed at the instance level in which the **stateMachine** is defined.
- The merging equation assigns the inner variable the value of the corresponding auxiliary variable of the currently active state in the following form:

```

x = if activeState(a) then $a.x
    elseif activeState(b) then $b.x
    else previous(x);
    
```

Further on, it is possible to improve the comprehensibility of the state machine representation by collecting all transitions and merging equations associated to a **stateMachine** section within that associated section (note that this is a pretty-print consideration and not a requirement for giving an unambiguous semantics).

4 Implementation

The proposed flattening for state machines has been implemented in the OpenModelica compiler. The process is depicted in Figure 4. State machines are first flattened in

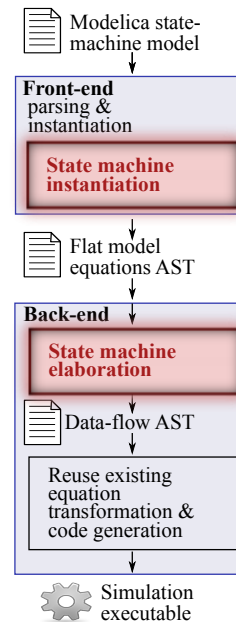


Figure 4. Outline of the state machine compilation process.

the compiler front-end according to the approach that was outlined in Section 3. After that, the state machine structures are further elaborated in the back-end where they are translated to basic data-flow equations (by transforming the abstract syntax tree (AST)). The translation to data-flow equations is inspired by the state machine compilation approach described by Colaço et al. (2005) and is a fairly direct encoding of the equations provided in the *Semantics Summary* of (Modelica Association, 2012, Section 17.3.4).

The current back-end implementation does not lead to very efficient code, *e.g.*, translation of the simple state machine example from Listing 1 to data-flow equations leads to 24 (mostly **Boolean**) data-flow variables (and equations) in the back-end (see Appendix B). However, this can be optimized to produce fewer variables and equations in future versions of the compiler without having to change the underlying symbolic representation produced by the front-end. The advantage of the current back-end implementation is the possibility to reuse most of the existing equation transformation and code generation facilities without further modification.

The current prototype needs a workaround to compensate for the not yet implemented support for Modelica’s clocked synchronous language extension (Modelica Association, 2012, Chapter 16). The “hack” in the back-end is to wrap all state machine related equations in a **when**-equation with a sampling period of one second and replace

all `previous(...)` operators by `pre(...)` operators, similarly to following code snippet:

```
when sample(0.0, 1.0) then
  i = if smOf.state1.activeState == 2
      then -1 + pre(i)
      else 2 + pre(i);
end when;
```

This restriction can be lifted easily as soon as the clocked synchronous language elements are fully supported by our compiler. Meanwhile the workaround allows to experiment with state machine implementations in parallel and independently to ongoing work related to synchronous languages elements support.

5 Conclusion

This paper proposed a dedicated symbolic representation for flattened Modelica state machines. The representation explicitly preserves crucial structural and relational information in the human readable flat Modelica representation. Hence, the proposed representation avoids ambiguities and can be interpreted straightforwardly by human inspection. This is in contrast to the ambiguous and hard to interpret flat representations of state machine models which are generated by existing tools. Furthermore, this representation is well suited for further computational processing in the back-end, because it becomes unnecessary to elaborately re-construct important structural information solely from the basic data-flow equations that are typically available at that later compiler phase.

At the same time the proposed representation strives to avoid an early commitment to implementation details for the specified state machine logic, *i.e.*, it refrains from performing a full translation of the state machine constructs to basic clocked synchronous data-flow equations in the flattened representation. In that way it allows one to postpone implementation decisions, that would potentially hinder code optimization techniques, to later translation stages.

The approach has been implemented in the OpenModelica compiler and successfully tested on a number of state machine models.

Acknowledgements

This work has been supported by Vinnova in the ITEA2 MODRIO project, by EU in the INTO-CPS project, and by the Swedish Government in the ELLIIT project. The Open Source Modelica Consortium supports the OpenModelica project.

References

R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and ver-

ification of hybrid systems. *Hybrid systems*, 736:209–229, 1993.

Marc Bouissou, Hilding Elmqvist, Martin Otter, and Albert Benveniste. Efficient Monte Carlo simulation of stochastic hybrid systems. In Hubertus Tummescheit and Karl-Erik Årzén, editors, 10th *Int. Modelica Conference*, Lund, Sweden, March 2014. doi:10.3384/ecp14096715.

Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet. A conservative extension of synchronous data-flow with state machines. In *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, pages 173–182, New York, NY, USA, 2005. ACM. ISBN 1-59593-091-4. doi:10.1145/1086228.1086261.

Hilding Elmqvist, Fabien Gaucher, Sven Erik Mattsson, and Francois Dupont. State Machines in Modelica. In Martin Otter and Dirk Zimmer, editors, 9th *Int. Modelica Conference*, Munich, Germany, September 2012. doi:10.3384/ecp1207637.

Hilding Elmqvist, Sven Erik Mattsson, and Martin Otter. Modelica extensions for Multi-Mode DAE Systems. In Hubertus Tummescheit and Karl-Erik Årzén, editors, 10th *Int. Modelica Conference*, Lund, Sweden, May 2014. doi:10.3384/ECP14096183.

Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley IEEE Press, 2014. ISBN 9781-118-859124.

David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987. ISSN 0167-6423. doi:10.1016/0167-6423(87)90035-9.

Florence Maraninchi and Yann Rémond. Mode-Automata: a new domain-specific construct for the development of safe critical systems. *Science of Computer Programming*, 46:219–254, 2003.

Modelica Association. Modelica—A Unified Object-Oriented Language for Systems Modeling v3.3. Standard Specification, May 2012. Available at <http://www.modelica.org/>.

Marc Pouzet. *Lucid Sychrone Tutorial and Reference Manual*, 2006.

Paulo Tabuada. *Verification and Control of Hybrid Systems A Symbolic Approach*. Springer US, 2009. doi:10.1007/978-1-4419-0224-5.

Bernhard Thiele. State Machines in OpenModelica - Current Status and Further Development. In *OpenModelica Annual Workshop*, Linköping, Sweden, 2. February 2015. Open Source Modelica Consortium (OSMC) and Linköping University (LiU). URL <http://www.modprod.liu.se/openmodelica-2015?l=en>.

A Flat Modelica for the Hierarchical State Machine Example

The complete listing of the flat Modelica representation of the hierarchical state machine example from Section 3.2.

```

class HierarchicalSM
  Integer x(start = 0);
  Integer z(start = 0);
  Integer y(start = 0);
  input Boolean i = true;
  input Boolean j = false;
  stateMachine smOf.a
  state a
    Integer $a.y;
    Integer $a.z;
    Integer $a.x;
    stateMachine smOf.a.c
    state a.c
      Integer $a.c.y;
      equation
        $a.c.y = 1 + previous(y);
      end a.c;

    state a.d
      Integer $a.d.y;
      equation
        $a.d.y = -1 + previous(y);
      end d;
    end smOf.a.c;

  stateMachine smOf.a.e
  state a.e
    Integer $a.e.z;
    equation
      $a.e.z = previous(z) + y;
    end a.e;

  state a.f
    Integer $a.f.z;
    equation
      $a.f.z = previous(z) - y;
    end a.f;
  end smOf.a.e;

equation
  $a.z = if activeState(a.e) then $a.e.z
        elseif activeState(a.f) then $a.f.z
        else previous(z);
  $a.y = if activeState(a.c) then $a.c.y
        elseif activeState(a.d) then $a.d.y
        else previous(y);
  initialState(a.e);
  transition(a.e, a.f, $a.z > 100,
    false, true, false, 1);
  transition(a.f, a.e, $a.z < 50,
    false, true, false, 1);
  transition(a.c, a.d, $a.y == 10,
    false, true, false, 1);
  transition(a.d, a.c, $a.y == 0,
    false, true, false, 1);
  $a.x = 1 + previous(x);
  initialState(a.c);
end a;

state b
  Integer $b.x;
equation
  $b.x = -1 + previous(x);
end b;
end smOf.a;

```

```

equation
  z = if activeState(a) then $a.z
      else previous(z);
  x = if activeState(a) then $a.x
      elseif activeState(b) then $b.x
      else previous(x);
  transition(a, b, z > 100,
    false, true, false, 1);
  transition(b, a, x == 0,
    false, true, false, 1);
  initialState(a);
end HierarchicalSM;

```

B Back-End Equations

The state machine elaboration in the back-end translates the state machine representation from the front-end to basic data-flow equations (see Figure 4). The intermediate system of equations can be retrieved from the back-end by using debugging functions. The listing below shows the equation system which is generated for the simple state machine example from Listing 1. For better readability the debug output has been reformatted to resemble the typical flat Modelica style. Obviously, the behaviour of the simple state machine can already be described by a fraction of the actually generated equations. However, such optimizations are not performed in the current prototype implementation.

```

model SimpleSM
// parameters
Integer smOf.state1.tPriority[2] = 1
Boolean smOf.state1.tSynchronize[2] = false
Boolean smOf.state1.tReset[2] = true
Boolean smOf.state1.tImmediate[2] = false
Integer smOf.state1.tTo[2] = 1
Integer smOf.state1.tFrom[2] = 2
Integer smOf.state1.tPriority[1] = 1
Boolean smOf.state1.tSynchronize[1] = false
Boolean smOf.state1.tReset[1] = true
Boolean smOf.state1.tImmediate[1] = false
Integer smOf.state1.tTo[1] = 2
Integer smOf.state1.tFrom[1] = 1
Integer smOf.state1.nState = 2
// variables
Integer i(start=0);
Boolean state2._active;
Boolean state1._active;
Boolean smOf._state1._init(start=true);
Boolean smOf._state1._stateMachineInFinalState;
Boolean smOf._state1._finalStates[2];
Boolean smOf._state1._finalStates[1];
Boolean smOf._state1._nextResetStates[2];
Boolean smOf._state1._nextResetStates[1];
Boolean smOf._state1._activeResetStates[2];
Boolean smOf._state1._activeResetStates[1];
Boolean smOf._state1._nextReset;
Integer smOf._state1._nextState;
Boolean smOf._state1._activeReset;
Integer smOf._state1._activeState;
Integer smOf._state1._fired;
Boolean smOf._state1._selectedReset;
Integer smOf._state1._selectedState;
Boolean smOf._state1._reset;
Boolean smOf._state1._active;
Boolean smOf._state1._cImmediate[2];
Boolean smOf._state1._c[2];
Boolean smOf._state1._cImmediate[1];

```

```

Boolean smOf._state1._c[1];
equation
when {sample(1.0, 1.0), initial()} then
  state2.active = smOf.state1.active
  and smOf.state1.activeState == 2;
state1.active = smOf.state1.active
  and smOf.state1.activeState == 1;
smOf.state1.active = true;
smOf.state1.reset = pre(smOf.state1.init);
smOf.state1.init = false;
smOf.state1.stateMachineInFinalState =
  smOf.state1.finalStates[
    smOf.state1.activeState];
smOf.state1.finalStates[2] = max(
  if smOf.state1.tFrom[2] == 2 then 1 else 0,
  if smOf.state1.tFrom[1] == 2 then 1 else 0
) == 0;
smOf.state1.finalStates[1] = max(
  if smOf.state1.tFrom[2] == 1 then 1 else 0,
  if smOf.state1.tFrom[1] == 1 then 1 else 0
) == 0;
smOf.state1.nextResetStates[2] =
  if smOf.state1.active then
    if smOf.state1.selectedState == 2 then false
    else smOf.state1.activeResetStates[2]
  else pre(smOf.state1.nextResetStates[2]);
smOf.state1.nextResetStates[1] =
  if smOf.state1.active then
    if smOf.state1.selectedState == 1 then false
    else smOf.state1.activeResetStates[1]
  else pre(smOf.state1.nextResetStates[1]);
smOf.state1.activeResetStates[2] =
  if smOf.state1.reset then true
  else pre(smOf.state1.nextResetStates[2]);
smOf.state1.activeResetStates[1] =
  if smOf.state1.reset then true
  else pre(smOf.state1.nextResetStates[1]);
smOf.state1.nextReset = if smOf.state1.active
  then false
  else pre(smOf.state1.nextReset);
smOf.state1.nextState =
  if smOf.state1.active
  then smOf.state1.activeState
  else pre(smOf.state1.nextState);
smOf.state1.activeReset =
  if smOf.state1.reset then true
  else
    if smOf.state1.fired > 0 then
      smOf.state1.tReset[smOf.state1.fired]
    else smOf.state1.selectedReset;
smOf.state1.activeState =
  if smOf.state1.reset then 1
  else
    if smOf.state1.fired > 0 then
      smOf.state1.tTo[smOf.state1.fired]
    else smOf.state1.selectedState;
smOf.state1.fired = max(
  if
    if smOf.state1.tFrom[2] ==
      smOf.state1.selectedState then
      smOf.state1.c[2]
    else false
      then 2 else 0,
  if
    if smOf.state1.tFrom[1] ==
      smOf.state1.selectedState then
      smOf.state1.c[1]
    else false
      then 1 else 0);
smOf.state1.selectedReset =
  if smOf.state1.reset then true
  else pre(smOf.state1.nextReset);
smOf.state1.selectedState =
  if smOf.state1.reset then 1
  else pre(smOf.state1.nextState);
smOf.state1.c[2] =
  pre(smOf.state1.cImmediate[2]);
smOf.state1.cImmediate[2] = i < 1;
smOf.state1.c[1] =
  pre(smOf.state1.cImmediate[1]);
smOf.state1.cImmediate[1] = i > 10;
i = if smOf.state1.activeState == 2
  and smOf.state1.active then -1 + pre(i)
  else if smOf.state1.activeState == 1
  and smOf.state1.active then 2 + pre(i)
  else pre(i);
end when;
end SimpleSM;

```


Exploiting Repeated Structures and Vectorization in Modelica

Joseph Schuchart¹ Volker Waurich² Martin Flehmig¹
Marcus Walther¹ Wolfgang E. Nagel¹ Ines Gubsch²

¹Center for Information Services and High Performance Computing, TU Dresden, Germany

²Chair of Construction Machines and Conveying Technology, TU Dresden, Germany ,

{forename.surname}@tu-dresden.de

Abstract

Large and highly-detailed Modelica models are frequently modeled by utilizing repeated structures, which is a repetition of various elements that are linked together in an iterative manner. While the Modelica language standard supports the representation of repeated structures, most Modelica compilers do not exploit their advantages for efficient simulations. Instead, all repeated equations are flattened and all array variables are expanded. This leads to unnecessarily long compile times and higher memory consumption. Another aspect that has been yet inadequately considered and is closely connected to repeated structures is vectorization. The vector units of modern CPUs can be engaged to perform SIMD (Single Instruction, Multiple Data) operations, executing the same instruction on multiple data points in parallel. This reveals a high potential for faster simulations. This paper discusses the advantages of utilizing repeated structures for modeling in order to achieve both faster compilation and simulation times. The potentials of preserving for loops throughout compilation are demonstrated using a basic implementation in the OpenModelica Compiler. The effect on the simulation time by enabling vectorization is demonstrated for an appropriate model.

Keywords: SIMD, Vectorization, OpenModelica, Translation, Repetitions

1 Introduction and Related Work

The Modelica language is capable of describing large models with few lines of code by using repetitions of submodules. In general, submodels can be connected in an iterative manner by using `for` loops to express the repeated model structure. Models that contain repetitions are common in physical and technical modeling, e.g., models of battery packs and chain gears. Repeated structures can also be introduced by discretizing model elements, e.g., electrical wires or hydraulic pipes. Figure 1 represents a discretized model of an electrical wire. The underlying model is a distributed RC-model which

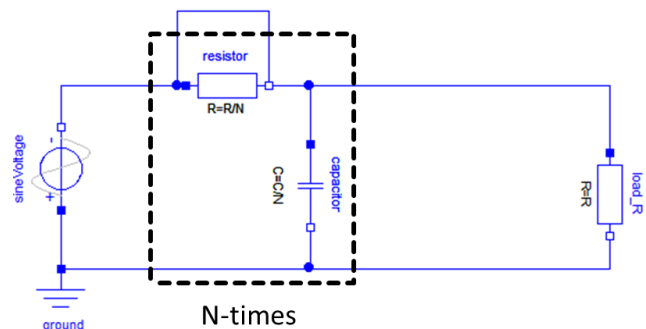


Figure 1. Distributed RC-model of an electrical wire using N repeated model elements.

is typically used to describe transmission behavior in electrical circuits. The circuit of resistor and capacitor is repeated N -times. Hence, all equations and variables which are defined in the resistor and capacitor model will appear N -times in the DAE system. Modelica compilers usually flatten the repeated equations without utilizing the information on repetitions. This can be explained by the fact, that the algorithms which transform the acausal DAE system into a causal, solvable ODE system have not been adopted to make use of repeated structures. Therefore, the symbolic manipulation handles an unnecessarily large amount of equations and variables and – as a consequence – the generated code does not contain any `for` loops anymore.

The potential of preserving repeated modules has already been illustrated (Zimmer, 2009). Zimmer describes the obvious potential of aggregating repeated modules and gives a basic method to approach the problem. He also highlights further issues regarding the symbolic algorithms in the compiler backend as will be explained later. In contrast to this work, the present paper will focus on iterated repetitions provided by `for` loops rather than by detecting modules. Another approach for addressing the topic has been proposed by Höger (Höger, 2011). By means of separate compilation of Modelica source code and subsequent linking, precompiled partial models can lead to smaller programs. Höger explains

the benefits of compiling source code before flattening the model but indicates problems related to the symbolic manipulation.

As has been stated already, symbolic manipulation for attaining a causal model needs further considerations. The main issues are related to index-reduction, typically performed by the Pantelides algorithm (Pantelides, 1988) and ordering of all equations and variables in a block-lower-triangular form, typically done by using the Tarjan algorithm (Tarjan, 1972). Arzt has provided a basic approach to adapt the Pantelides algorithm in order to exploit repeated structures (Arzt et al., 2014). Repeated equations and variables are collected in repetitive modules which share the same matching and derivation pattern. A prototypic implementation proved the ability to keep compilation time constant but has not yet been introduced into a usable Modelica compiler.

Preserving repeated structures throughout symbolic transformation can lead to decreased compilation times and memory consumption. It also reveals the potential for exploiting the vector units of a CPU efficiently by providing the underlying compiler with opportunities for creating vectorized code.

The remainder of the paper is structured as follows: In Section 2, a basic implementation for preserving repeated structures through the backend will be illustrated and the benefits of modeling `for` loops in causal sections are presented in Section 3. Section 4 describes the means and requirements to exploit vector computation and presents the impact on the simulation time of an example model. Finally, conclusions and an outlook are contained in Section 5.

2 Preserving Repeated Structures Throughout Compilation

Acausal Modelica models have to be transformed symbolically to attain a solvable, causal differential-algebraic system of equations (i.e., DAE system). Thus, every variable has to be assigned to a specific equation and a computation order has to be determined. The assignment between equations and variables is called matching and the ordering in a sequence of single equations and algebraic loops (i.e., systems of equations) is typically done by Tarjan's strongly connected component algorithm. If necessary, the index of the DAE system has to be reduced by performing the Pantelides algorithm. Besides these mandatory manipulations, there are even more Modelica-specific algorithms to improve the computable model, e.g., removal of trivial equations or tearing methods. These algorithms are well probed and standard in Modelica model compilation but only a few of them have been extended to handle repeated objects like the Modelica `for` loops. The mentioned algorithms can be applied easily to expanded `for` loops and array vari-

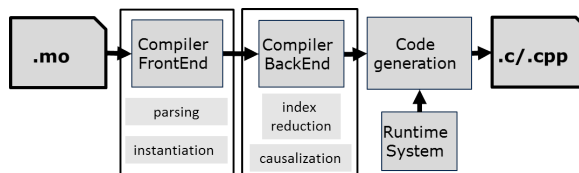


Figure 2. Compilation process in the OpenModelica Compiler.

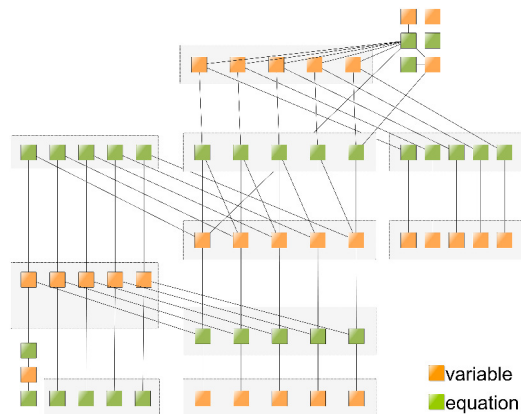


Figure 3. Flat model representation as a bipartite graph. Green squares represent equations and orange squares represent variables. Squares in a grey box can be gathered in a compact `for` loop notation or as an array-variable. Two vertices have a connecting edge if the variable appears in the equation. A removing of trivial equations has already been applied.

ables as it is the current default implementation in the OpenModelica Compiler.

The workflow of the compilation process in the OpenModelica Compiler is depicted in Figure 2. After parsing the `mo`-file, all model elements are instantiated and a list of equations and variables, the flat DAE, is generated. This flat DAE representation is the basis for all further manipulations like index-reduction, causalization, and additional optimizations. The compiler backend creates a solvable model which is used to generate C or C++ code in order to compile an executable. If the `for` equations in the Modelica model should be used throughout the compilation procedure, the instantiation has to output a valid representation of the iterated equations for the repeated model elements. This is ongoing work within the frontend development and therefore is not covered by the presented paper.

In order to demonstrate the effects of retaining loops throughout compilation, a basic implementation in the compiler backend has been created. The instantiation still generates a flat DAE. However, the compiler backend now collects the flattened equations and array variables and establishes compacted `for` equations by comparing the terms of equations in order to find similarities. If a continuous iteration can be identified, an equivalent `for` equation is set up. For the wire model presented in Figure 1, the bipartite graph depicted in Figure 3 illustrates the compact notation of `for` equations.

As can be seen, the bipartite graph in Figure 3 has a repetitive structure. For this model, the number of repetitions is 5. The same structure can be expressed using the compact notation of `for` loops. There are two kinds of `for` loop equations that are passed through the compilation process. On the one hand, there is the repetition of several equations with differently indexed variables:

```

for i in 1:5 loop
  resistor[i].LossPower =
    resistor[i].v * resistor[i].i;
end for;

```

On the other hand, it is possible to have a single equation with a repetition of terms, e.g., in a summation that is represented by the `sum`-operator, for example:

```

ground.p.i + sineVoltage.p.i + load_R.n.i
+ sum((i->1:5) + capacitor[i].n.i) = 0.0

```

2.1 Symbolic Transformation of Repeated Structures

The following section outlines the procedure of preserving `for` equations throughout model compilation. It basically covers the removal of trivial equations, matching, and identification of the computation sequence. The implemented prototype is only able to handle systems of maximum index one and without algebraic loops. Since this is a hard restriction, the fallback solution is to scalarize the DAE-system completely at any stage of compilation. At least the removal of trivial equations can benefit from the compact notation in that case. The results will include a benchmark for the removal of trivial equations exploiting `for` equations.

Since the information about repeated model elements is currently not available from instantiation, it has to be collected from the completely flattened DAE system.

To collect equations in `for` constructs, all flat equations have to be filtered for equations containing iterated variables. If an equation contains solely iterated variables, other equations which share the same algebraic terms without considering the array indexes are gathered as a `for` loop. Among these equations, the array indexes are compared and examined to determine the start and end index of the iteration. A linear iteration with a step of one is assumed. Equations containing several iterated instances of the same array variable are checked for possible terms like a summation of iterated variables. Since this process can be avoided by instantiating `for` equations and unexpanded arrays directly, the collection of `for` equations will not be taken into account for subsequent benchmarks.

Hence, the basis for the presented method is a flat DAE including both scalar variables and unexpanded array-variables which have to be matched to either single equations or `for` equations. Figure 4 depicts an overview of the procedure.

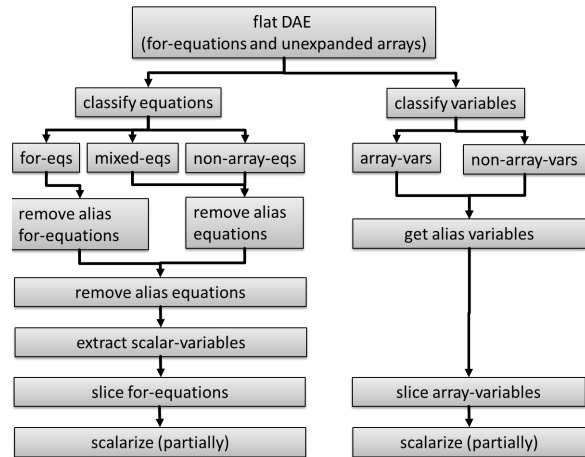


Figure 4. Overview of the process of collecting `for` equations.

Classify equations: In order to reduce the problem size, all DAE-equations are classified into the three groups *for-equations* (containing only unexpanded array-variables), *non-array-equations* (containing only non array-variables), and *mixed-equations* (containing scalar array-variables). Variables are classified as either array-variables or non-array-variables.

Removal of trivial equations: The removal of trivial equations is comprised of both the removal of simple equations like $a = b$, $a = -b$, or $a = \text{const}$ and a replacement of the respective alias variables to maintain the balance of equations and variables. This is a common optimization in model compilation in order to reduce the system size. Trivial equations have a prominent fraction among all model equations (from 44% to 73%) so this optimization is an eminent operation for an efficient model compilation. The scalar implementation for removing of trivial equations is applied on the *mixed* and *non-array* equations. Trivial equations do occur in *for-equations* as well and the removal and replacement of alias variables can be adopted. The scalar implementation has to find N trivial equations for N repeated equations whereas the implementation for the compact `for` equations only has to detect a single trivial assignment to assign N alias variables.

Partial slicing of ranges: The goal of the upcoming matching is to assign variables explicitly to all equations. It is settled that a scalar equation can only solve a scalar variable and an N -dimensional `for` equation can only solve an N -dimensional array-variable. Therefore, it has to be ensured that every `for` equation is connected to array-variables of the same dimension range. Besides that, it is possible, that a certain scalar variable is solved in one of the *mixed-equations* or that it has to be replaced by its alias variable. Hence, the unexpanded array-variables have to be sliced in ranges of their com-

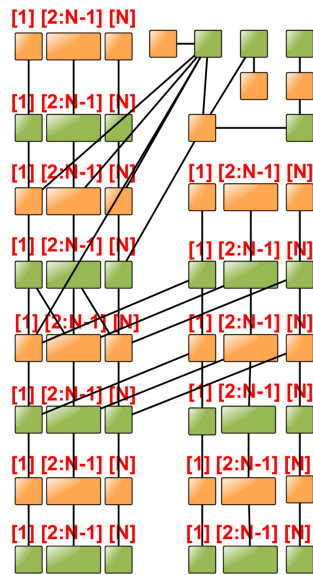


Figure 5. Bipartite graph with balanced dimensions.

plete dimension and single scalar array elements. First of all, the scalar array variables which have been replaced and the scalar array variables which occur in the *mixed-equations* have to be sliced from the unexpanded array. As a second step, a balance of dimension ranges between `for` equations and adjacent array-variables has to be established in the bipartite graph. This is performed by traversing all array-variables and compare the dimensions of the adjacent `for` equations. It has to be checked whether the dimensions have to be adjusted by slicing scalar equations or variables. Figure 5 depicts the partially sliced system of the wire model. As can be seen, every N -dimensional equation is connected to N -dimensional array variables (or scalar variables which have to be matched to a scalar equation instance).

Matching: If the bipartite graph is completely balanced with respect to the dimensions, matching can be performed as usual. Every equation node with only one adjacent variable node will be matched to this particular variable node. Thus, a matched variable can be taken as known and all edges can be removed from the graph. The order of matching assignments corresponds to the computation sequence. The current implementation offers a basic compilation of models containing repetitions. At the moment, no index-reduction algorithm is realized and no algebraic loops are allowed to occur. Furthermore, interruptions in the iteration space are not yet handled. However, these features are usually not required in electrical and hydraulical transmission elements. When compiling the electrical wire model, an ODE system is generated which contains a constant number of equations. Only the number of iterations inside the `for` loops is dependent on the number of discretized elements.

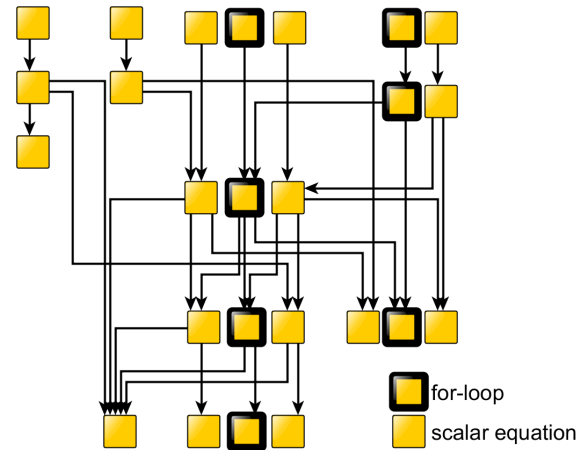


Figure 6. Task graph representation of causal ODE system with compact `for`-equation notation.

2.2 Results

Figure 6 depicts the task-graph of the causalized system for the wire model. A task graph represents the computation sequence of equations and algebraic loops which are displayed as nodes connected by edges referring to data dependencies (Walther et al., 2014). A node depicts a single equation to be solved in order to compute the successive equations. A bold task represents a vector task containing a `for` loop.

As can be seen, the compact notation is preserved throughout symbolic transformation. This results in faster compilation due to a reduced problem size. Figure 7 shows the number of equations which have to be processed in the compiler. By using the compact `for` notation, the number of equations can be kept at a constant size for different numbers of repetitions. Applying the removal of trivial `for` equations and subsequent scalarization of the DAE reduces the problem size significantly, as is shown with the green bars.

Figure 8 shows the time to translate the model from a flattened DAE-system to a solvable system. Obviously, the symbolic transformation for the default compilation of a flattened DAE-system does not scale linearly. The translation using the compact `for` notation results in a nearly constant compilation time. In order to reveal the advantages by using `for` notation for the removal of trivial equations, a third bar is depicted. The green bars show the compilation time if the removal of trivial equations has been performed on the `for` equations and the system is scalarized completely afterwards. This can be considered the minimal solution to exploit repeated equations.

The task graph representation is the basis for the generation of program code. Like in Modelica, `for` loops are a first level language concept in C and C++. By adopting the code generation to the new `for` loop tasks, the size of the generated code and the compiled executable are reduced, as depicted in Figure 9.

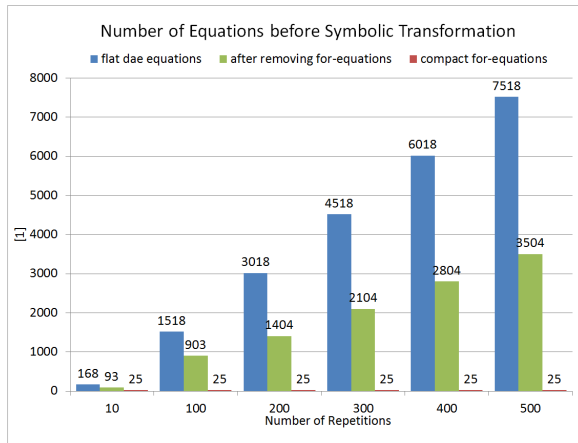


Figure 7. Comparison of number of equations to process the symbolic transformation for the default implementation, the vectorized compilation and completely scalarized system after removing of trivial `for` equations.

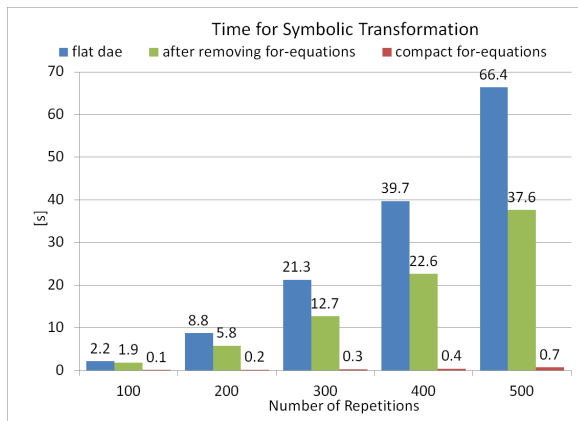


Figure 8. Comparison of symbolic transformation time between default implementation, complete vectorized compilation and completely scalarized system after removing of trivial `for` equations.

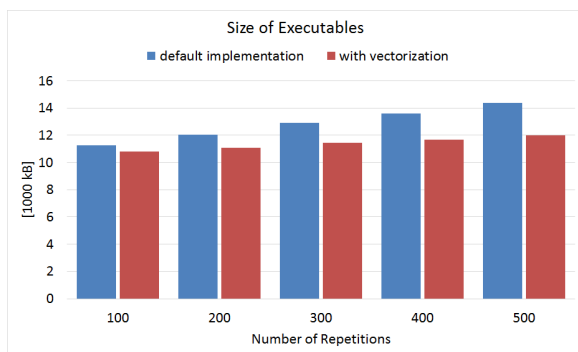


Figure 9. Comparison of executable size between default implementation and vectorized compilation.

The difference in size of the executables is little since the declaration of system equations is only one small part of the overall program code. Nevertheless, it is an evident implication of the compact `for` loop notation.

The wire model does not contain algebraic loops or complex function calls. Hence, it can be simulated very fast which is why no simulation time comparison has been done. The benefit of vectorization for improving simulation times will be presented in the Section 4.

3 Causal For Loop Statements in Modelica

Loop statements in Modelica models can both occur in algorithm and equation sections. In algorithm sections like Modelica functions, they are trivially passed through the model compilation since they do not have to be causalized or manipulated. Therefore, causal `for` loop statements can be forwarded directly to the code generation and vectorization can be applied. This can lead to improved simulation performance so using `for` loops should be a common modeling best practice.

One exemplary model is `Fluid.Examples.BranchingDynamicPipes` from the Modelica Standard Library 3.2.1. This model makes heavy use of property computations defined in the Media library. Various Modelica functions are defined that compute the property values for the employed medium. One function used when simulating moist air is `Media.Air.MoistAir.saturationPressureLiquid`. The default implementation to calculate the saturation pressure is:

```
psat := exp(((
  a[1]*r1^n[1]
+ a[2]*r1^n[2]
+ a[3]*r1^n[3]
+ a[4]*r1^n[4]
+ a[5]*r1^n[5]
+ a[6]*r1^n[6])
* Tcritical) / Tsat) * pcritical;
```

Since a major part of this expression consists of operations on contiguous memory elements, this computation can be vectorized. The function can be rewritten to compute the saturation by means of a `for` loop:

```
for i in 1:6 loop
  aux := aux + a[i]*r1^n[i];
end for;
psat := exp((aux * Tcritical) / Tsat)
* pcritical;
```

This is just one example of how Modelica functions can utilize `for` statements. Due to the extensive use of `for` statements in this model, vectorization can have distinct effects on simulation performance, as will be explained in the following section.

4 Vectorization

In many cases, modern CPU architectures provide vector units that allow parallel execution of the same in-

struction on multiple data elements which is known as SIMD (Single Instruction, Multiple Data). SIMD parallelization often requires less hardware for parallel computation than multiple full cores do. Hence, if used correctly, the exploitation of available SIMD instruction sets promises improved performance without requiring additional hardware. It also improves energy-efficiency, since more computation can be done per clock cycle and thus per Watt.

There are various ways of generating vector code. While low-level techniques, like vector intrinsics and assembly code (Intel), could in theory be employed in the code generation of a Modelica compiler backend, it requires much more caution and effort by the developers of the Modelica compilers. Recent developments on the C/C++ compiler infrastructures have brought powerful tools to developers that make it easier to exploit the computational power of vector units. Moreover, relying on the compiler to efficiently vectorize code also guarantees compatibility with future hardware, which otherwise would require extensions to the Modelica compiler in order to support new hardware platforms. Thus, the presented work focuses on automatic compiler vectorization in order to ensure correctness, portability, and ease of maintenance.

4.1 Compiler-based Vectorization

Vectorization is based on the SIMD principle, which requires the same instruction to be executed on different data points. This is usually the case, if the application makes use of loop control structures that iterate over a vector of data points. Modern compilers, e.g., the GNU compiler collection (GCC), the Intel compiler, and other compiler products and frameworks, incorporate sophisticated analysis and optimization logic to detect vectorizable loops and emit the respective instructions for the available hardware (Maleki et al., 2011). However, certain constraints have to be met in order to allow the compiler to correctly and efficiently vectorize a loop (Cordeu, 2012), including:

Absence of data dependencies: An iteration N of the vectorized loop may not consume the result of a previous iteration $N - 1$ (read-after-write dependency). The same holds true for write-after-read dependencies, although modern compilers apply heuristics to detect vectorizable patterns such as reductions, e.g., in the example code in Section 3.

Aligned memory accesses: Modern CPU and memory architectures provide access to memory through caches, with the cache line size commonly at 64 B, which are loaded at once into the cache. Using non-unit-stride memory accesses might require loading multiple cache lines, which in turn could impact performance and thus outweigh the benefits of vectorization.

Linear iteration space and known step size: in order to correctly transform loops into vector statements, the step size and trip count have to be known at least at runtime. This excludes infinite loops, loops that are not bound by an explicit index variable, as well as loops with early exits.

Restricted function calls: Calling functions inside loops prevents the compiler from properly vectorizing the loop, unless these functions can be transformed into vectorized code, e.g., transcendental functions such as `pow()` or `exp()`, or functions that have been otherwise marked as vectorizable, e.g., using OpenMP SIMD statements (OpenMP, Sect. 2.8.2).

Avoiding branches: Branches commonly create conditionally diverging code paths which cannot be vectorized easily. With today's larger vector units, the compiler might be able to use mask registers to restrict operations to parts of a vector. However, branches should be avoided in general for best performance.

4.2 Modern Vector Architectures

Modern x86 CPUs are equipped with 128 up to 256-bit wide vector units and support for the Advanced Vector eXtensions (AVX and AVX2), which enable the parallel computation of up to four double or eight single precision floating point values. Other architectures, e.g., the ARM architecture with its NEON instruction set, also support vector instructions.¹ Without actually using these instructions, developers only make use of a fraction of the theoretical peak performance available. Upcoming generations of Intel CPUs will provide support for the AVX-512 instruction set, which once more doubles the width of the vector registers to 512 bit or eight parallel double precision floating point operations in parallel (Intel).

4.3 Results

As described in Section 3, several functions from the Modelica Standard Library that are used by the model `BranchingDynamicPipes` to contain vectorizable loops have been adjusted. These loops make heavy use of the `pow()` function, which has been vectorized by the compiler.

To compare the effects of vectorization on the runtime, we compiled the model into C++ code and translated it using different compiler settings (see Table 1). Since the modifications to the Modelica code were minor (combining unrolled loops into for statements with short trip

¹<http://www.arm.com/products/processors/technologies/neon.php>, accessed 2015-05-19

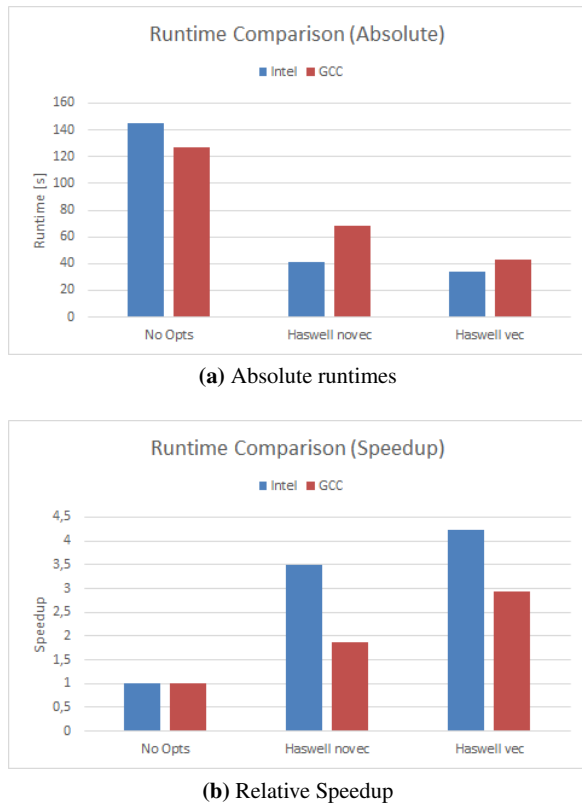


Figure 10. Resulting runtimes and speedup without optimizations, with standard compiler optimizations and with vectorization for different compilers.

counts), the behavior of the code did not change. The difference in performance between a loop and its unrolled counterpart are negligible if the loop body is computationally heavy, as is the case with the `pow()` function. Hence, changing the compiler flags that control the vectorization is sufficient for a fair comparison.

It should be noted that none of the compilers used actually perform vectorization if optimizations are disabled (`-O0`). Since this is the default configuration in OpenModelica, measurements for runs without any optimization are included, too.

The following measurements were performed on a single socket system equipped with 8 GB of RAM and an Intel Core i7-4770 (Haswell microarchitecture) eight core CPU with HyperThreading disabled. The CPU supports the AVX2 instruction set mentioned in Section 4.2. The system was running an up-to-date Ubuntu GNU/Linux version 14.04 with kernel 3.13.0-45-generic.

The differences in runtime between runs using no optimization, optimization level two (`-O2`) without vectorization and optimization level two with vectorization are presented in Figure 10a. The corresponding relative speedups are displayed in Figure 10b. The difference between a run without optimizations and with default optimizations is significant, ranging from 1.9 for the GNU compiler to 3.4 for the Intel compiler. By enabling vec-

torization, the code compiled with the GNU compiler provides a reduction in runtime of 37.5% on top of the default level two optimizations and a factor of 2.95 compared to the run without any optimizations. For the Intel compiler, the speedup gained by enabling level two optimizations is much more significant, around factor 3.5. However, the speedup from vectorization is only 20% on top of that, totaling to a factor of 4.23.

It should be noted that only a fraction of the code has been vectorized, namely the functions from the Media library that form the most compute intensive kernels in this model (see Section 3). On top of that it should be noted that for all optimized runs the FMA feature had to be disabled. Fused-Multiply-Add (FMA) is an instruction that combines one multiplication and one addition, avoiding rounding of intermediate results and potentially doubling the arithmetic throughput. However, it also has been observed to impact the numerical behavior of the application, potentially increasing run-times of the solver component.

5 Conclusion and Outlook

The presented work aims to motivate the use of `for` loops as a modeling best practice in Modelica. The benefits from exploiting these repeated structures have been demonstrated. Passing `for` loop constructs through the compilation process accelerates the symbolic transformation, the program compilation as well as the model execution. The size of the generated code and executable can be reduced. Furthermore, it also enables the utilization of vector-units to perform SIMD operations effectively. The simulation time can be reduced clearly when applying automatic vectorization on `for` loops.

The demonstrated loop-preserving compilation features are still limited in their functionality. First of all, an instantiation providing compact `for` equations would replace the costly collection of similar equations which is currently performed in the OpenModelica Compiler backend. In order to extend the range of manageable models, the implementation has to be extended to support index-reduction and algebraic loops. Also the interruption of repeated structures, nested-loops, or non-linear iterations are potential research topics. Moreover, loop fusion is another worthwhile research target that can provide higher arithmetic density by reusing intermediate values and improving vectorization efficiency.

As has been mentioned earlier, certain features of modern CPU instruction sets can lead to small numerical differences in the numerical behavior of the application, leading to significant consequences for the total simulation performance. Further investigations will be conducted to allow the full exploitation of available hardware features while avoiding increased simulation times.

	GCC	Intel
Version	4.9.2	15.0.3 20150407
Optimization Flags		
No Opts	-O0	-O0
Haswell novect	-O2 -fno-tree-loop-vectorize -march=haswell -mno-fma -ffast-math	-O2 -no-vec -no-simd
Haswell vect	-O2 -ftree-loop-vectorize -mno-fma -ffast-math -mveclibabi=svml -march=haswell	-O2 -xCORE-AVX2 -fno-fma

Table 1. Compiler versions and flags overview.

Acknowledgments

The presented work is part of the HPCOM project, that is funded by the Federal Ministry of Education and Research (BMBF) under the support code 01IH13002B. Parts of the work have been funded by the Intel Parallel Computing Center at the Center for Information Services and High Performance Computing at the TU Dresden.

References

- Matthias Arzt, Volker Waurich, and Jörg Wensch. Towards Utilizing Repeating Structures for Constant Time Compilation of Large Modelica Models. In *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT '14, pages 35–38, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666207. URL <http://doi.acm.org/10.1145/2666202.2666207>.
- Martyn Corden. Requirements for Vectorizable Loops, 2012. URL <https://software.intel.com/en-us/articles/requirements-for-vectorizable-loops>. Accessed 2015-05-19.
- Christoph Höger. Separate Compilation of Causalized Equations -Work in Progress. *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT 2011, Zurich, Switzerland, September 5, 2011, pages 113–120, 2011.
- Intel. *Intel Architecture Instruction Set Extensions Programming Reference*. Intel, October 2014. URL <https://software.intel.com/sites/default/files/managed/0d/53/319433-022.pdf>. Accessed 2015-05-19.
- Saeed Maleki, Yaoqing Gao, Maria J. Garzaran, Tommy Wong, and David A. Padua. An evaluation of vectorizing compilers. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 372–382, Oct 2011. doi:10.1109/PACT.2011.68. URL polaris.cs.uiuc.edu/~garzaran/doc/pact11.pdf.
- OpenMP. *OpenMP Application Program Interface*. OpenMP Architecture Review Board, Jul 2013. URL <http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf>.
- Constantinos C. Pantelides. The Consistent Initialization of Differential-Algebraic Systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988. doi:10.1137/0909014. URL <http://dx.doi.org/10.1137/0909014>.
- Robert E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, pages 146–160, 1972. URL langevin.univ-tln.fr/cours/PAA/extra/Tarjan-1972.pdf.
- Marcus Walther, Volker Waurich, Christian Schubert, Ines Gubsch, Andreas Hofmann, and Lars Mikelsons. Equation based parallelization of Modelica models. In *Proceedings of the 10th International Modelica Conference*, 2014.
- Dirk Zimmer. Module-Preserving Compilation of Modelica Models. *Proceedings of the 7th International Modelica Conference; Como; Italy; 20-22 September 2009*, (2):880–889, 2009.

High Fidelity Multibody Vehicle Dynamics Models for Driver-in-the-Loop Simulators

Mike Dempsey Garron Fish Juan Gabriel Delgado Beltran

Claytex Services Limited, UK, mike.dempsey@claytex.com

Abstract

Modern Driver-in-the-Loop simulators are sophisticated engineering tools that have been developed within Motorsport to support the development and optimization of race cars in Formula 1, NASCAR and Indycar. At the heart of the simulator is the vehicle model which has to accurately capture the behavior of the whole car. Modelica based vehicle models are used by many of the top teams because it enables a multi-domain vehicle model to be used in the simulators and support all the other simulation activities within the team. These technologies are now being deployed into road car applications which presents a number of additional challenges. One of the major differences is the need to include bushes within the suspension. This paper presents a number of the recent developments in Modelica based vehicle dynamics models for both Motorsport and road car applications including new suspension models with bushes, integration with tools to provide high fidelity LiDAR road data and real-time simulation of these models.

Keywords: Driver-in-the-Loop, vehicle dynamics, real-time, nonlinear bush models

1 Introduction

In the last few years driving simulators have been developed that can be used for more than basic procedural simulation such as driver training, evaluating human factors such as fatigue and stress, ergonomics and testing new man-machine-interfaces. The latest generation of systems make it possible to simulate a mathematical model of a car, over an exact replica of a road surface, with identical scenery and visual reference, with a *human* driver, in a safe, controlled, environment (Hoyle, 2014). These developments have been led by motorsport teams and organisations due to the restrictions in testing imposed by the governing bodies and the increasing complexity of the cars.

There are many technological developments that have enabled this including new software, new motion platforms, high fidelity real-time vehicle models and high precision LiDAR track data. LiDAR is an acronym for Light Detection and Ranging and typically this means that the whole track has been scanned with a laser to accurately measure the surface. In many cases the

motorsport organisations have developed their own in-house Driver-in-the-Loop (DiL) system, often integrating many different technologies coming from different suppliers (Toso, 2014). Some of these systems are being commercialised by the motorsport organisations to help them capitalise on the technological developments they have made in the development of these systems.

For Automotive OEM's the appeal of high fidelity driving simulators is that they can move the testing of new vehicle designs and parts into the virtual world and start the assessment of design decisions with professional drivers before committing to the production of a prototype. This approach also allows the design process to be accelerated because, for example, a change to a damper characteristic can be applied in seconds rather than having to wait while the mechanics strip and rebuild all 4 dampers to a new specification and refit them to the car.

This paper focuses on the recent enhancements in the vehicle models and the related interface to track data.

2 Integration with High Fidelity Road Data

2.1 Overview

rFpro have developed a tool called TerrainServer, that is capable of feeding 1cm resolution LiDAR data into a vehicle model at up to 5kHz, on standard PC hardware, enabling the vehicle model to be run in real-time. It was initially developed to support DiL simulators but it is equally capable of supporting offline simulation enabling the same track data to be used in both environments.

A Modelica library, also called TerrainServer, has been developed that provides an interface to the rFpro Terrain Server. This library enables Modelica based vehicle models to access high fidelity LiDAR data, and is compatible with the Modelica Standard Library and the Vehicle Dynamics Library.

The TerrainServer Modelica library provides a new tyre contact model, ground contact model, external functions to access rFpro Terrain Server and a new closed loop driver model. Figure 1 shows a full vehicle model, created using the Vehicle Dynamics Library enhanced with the TerrainServer interface and associated closed-loop driver model.

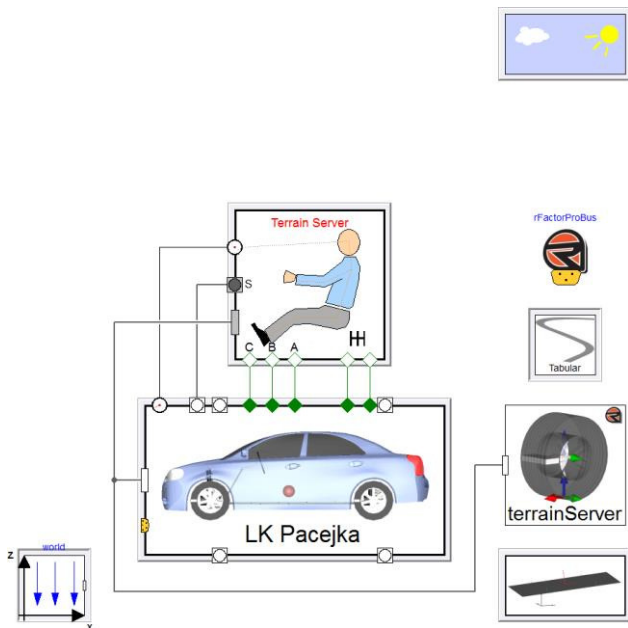


Figure 1: Vehicle model with TerrainServer interface

2.2 Tyre contact model

rFpro TerrainServer provides a number of different ways to use the LiDAR data with the most sophisticated tyre contact method referred to as a volumetric intersection sampling. Using this method the tyre is approximated as a cylinder and the resulting contact point is calculated by integrating the points within the tyre volume to return the contact patch centre and the averaged surface normal (rFactor Pro, 2014), see Figure 2. The returned data can be used in two different ways within the tyre contact model to suit different types of tyre model.

In method 1, the returned data is used to define the tyre contact point in the model. This can result in the surface normal not passing through the wheel centre and can also induce slip velocities due to the movement of

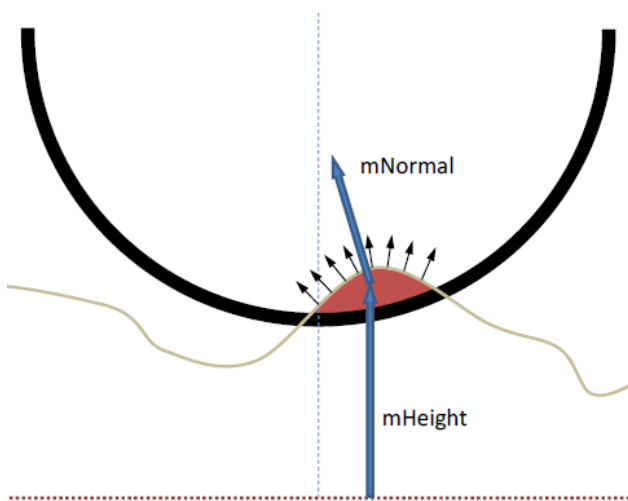


Figure 2: Calculation of the contact patch centre and average surface normal in rFpro TerrainServer

the contact patch centre as new data points enter and leave the tyre volume. As the spacing of the LiDAR data is reduced and the precision of the points improves this effect is reduced and provides an accurate contact point to the tyre model.

The problem with this approach is the interaction between the calculation of the contact point using the detailed road surface and the single point of contact tyre models typically used for handling simulation such as Pacejka. These tyre models work on the assumption that the road surface near the contact patch can be approximated by a flat plane and that the contact point lies within the tyre central plane (Pacejka 2012). This means that the smallest considered wavelength of the decomposed surface vertical profile is large with respect to the contact length and its amplitude small. The high fidelity track data used in rFpro TerrainServer provides data to the tyres that breaks this assumption, however a way to handle this has been developed.

In method 2, the returned data is used to define a plane underneath the wheel and the contact point is calculated as being the closest point to the wheel centre that lies within the ground plane and tyre central plane. This is illustrated in Figure 3 where the calculated plane and contact point are shown in green. This approach allows the assumptions in the single point of contact tyre models to remain valid, i.e. the road is treated as a flat plane underneath the tyre. This reduces the movement of the contact patch due to the entry and exit of points into the tyre volume and also ensures that the surface normal always passes through the wheel centre. The compromise in this approach is that the real surface detail available from the LiDAR data cannot be used to full effect by tyre models like the Pacejka model.

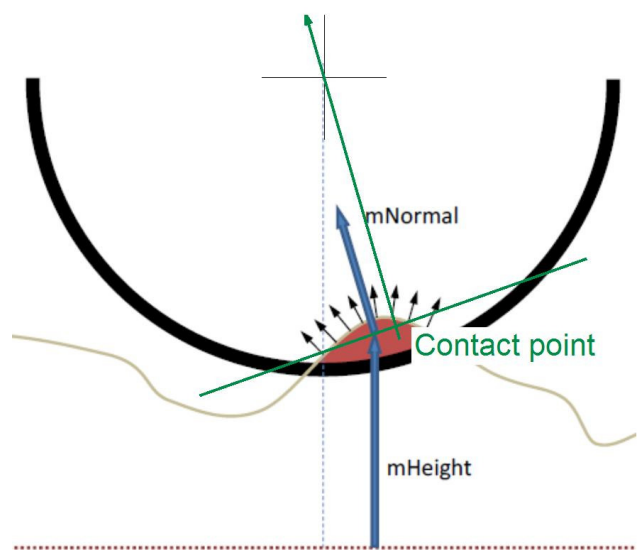


Figure 3: Calculation of the effective tyre contact point using method 2 where the rFpro data is used to define a plane underneath the tyre

3 Closed-loop driver model

3.1 Overview

A new closed-loop driver model has been developed for use with TerrainServer. The driver is a path following driver model with the trajectory defined in a new Path object that defines the driving line and speed profile to be followed. The driving line and speed profile would typically be captured from a session on the driving simulator with a professional driver, it can then be repeated and analysed offline to explore setup changes. This new driver model has several key differences to the existing model provided in the Vehicle Dynamics Library which include the way the target path is defined, how the preview points are determined and how the longitudinal tracking is implemented.

The path is defined as positions in the world coordinate system as a function of a distance along the path. A speed profile that is also a function of the distance along the path is included. The driver model looks at the path information to decide what steering, pedal and gear shift commands are necessary.

3.2 Path planning

The model uses 3 preview points: 2 are used in the longitudinal tracker and the lateral tracker can use either 1 point or average all 3. In the longitudinal tracker one point is used as the input to a PI controller to determine the throttle and brake pedal positions. This preview point should be close to the current driver position to achieve accurate tracking of the speed profile. The second point is used in a mode correction block to enable the driver model to anticipate a switch between acceleration and braking rather than waiting for the PI controller to respond after passing the transition point.

The lateral tracker calculates the angle from the vehicles current position and heading direction to the position of the preview point. From this angle it determines a steering angle that needs to be applied. Filters are used to limit the rate at which the driver can adjust the steering and pedals to keep the responses appropriate for the type of driver that is being represented.

The lateral tracking preview point is a variable distance ahead of the driver's current location and can be adjusted according to many factors including vehicle speed, lateral offset from the defined trajectory, curvature of the path and yaw velocity, see Figure 4. The preview distance and corresponding adjustments are implemented to give the driver model the ability to plan ahead and adjust the control strategy to suit the road and vehicle state just as a real driver does when driving the car.

The typical configuration of the driver model is that the preview point will move further ahead of the driver as the vehicle speed increases and this provides the basic preview distance. If the lateral tracking of the driver

model is not good, or the vehicle is unable to follow the path then the lateral offset will increase and the preview distance will also be increased. This is to avoid the driver trying to then turn too sharply when he is unable to follow the path. To avoid cutting sharp corners the preview distance is reduced as the curvature of the path increases. A large curvature value means a tight corner and to keep the lateral tracking performance within reasonable limits we make sure that the preview points are not too far ahead of the driver. The rate at which the preview point moves relative to the driver is limited to avoid large jumps in the position which would induce large, and unrealistic, changes in the steering command.

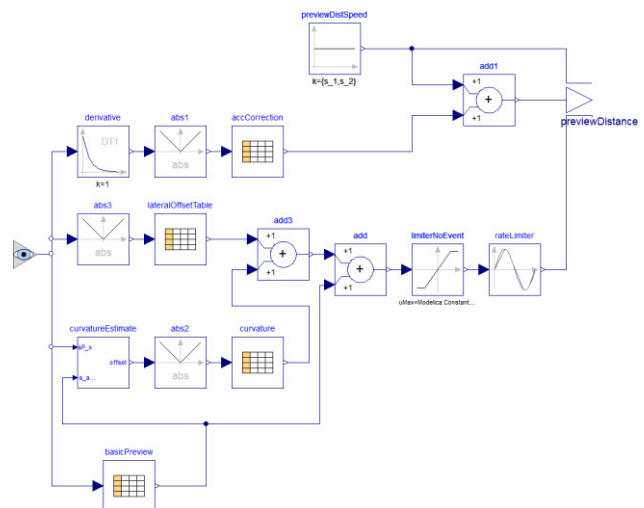


Figure 4: Preview distance calculation

3.3 Generating the target path

The path is generated by filtering the data recorded from the driving simulator to specify the minimum distance and the minimum time between points. This means that at low speeds the points that define the path will be at least the minimum distance apart but at higher speeds the points will be more spread out, for example at 50m/s with a minimum time of 0.2s the points will be 10m apart. This filtering is done to generate a smooth path for the driver model to follow.

Two ways of processing the recorded data are provided in the Modelica TerrainServer library. When using the Tabular path model the data is processed into a single continuous path that can be followed. There is also a Racing Lap path where the data is processed to extract an out lap and a single flying lap from the data. The start and end of the flying lap has to be blended together and blended with the out lap so that the flying lap can be looped allowing multiple laps to be simulated.

The driver model can also be exported and compiled as a model that can be run within the driving simulator environments. This enables an automatic driver to be used to verify the correct operation of the driving simulator platform and new vehicle models prior to a test with a human driver.

3.4 Exploring driver behavior

Using the various tuning parameters within the driver model we can explore how different driving styles influence the loads on the vehicle. For example by keeping the preview points close to the driver's position and having a high proportional gain and low integral gain in the longitudinal PI controller we can define a very aggressive driver that will work the steering wheel and pedals at a high frequency to follow the desired path very closely.

At the other extreme we can configure the driver so that the preview points are further ahead of the vehicle, and with a low proportional gain and high integral gain we get a much more relaxed driver behaviour but the path tracking performance will not be very good. For this relaxed driving profile we still need to keep 1 of the longitudinal preview points relatively close to the vehicle to ensure that he can still switch between acceleration and braking at the appropriate points and avoid large velocity overshoots which would not suit this style of driver behaviour.

Figure 5 shows a comparison of these two different driving styles on a short section of a test track. The aggressive driver has better lateral and longitudinal tracking than the relaxed driver, and achieves this through faster actuation of the pedals and steering wheel.

Through careful selection of the driver model parameters we have used this driver model in transient lapsim analysis. In these applications the driving line has been recorded from a simulator session with the professional driver, the use of the driver model then enables setup changes to be evaluated. This approach to lapsim analysis allows the full transient behaviour of the car to be considered which is not possible using quasi-static approaches.

4 Road car suspension models

4.1 Overview

A new set of suspension models has been developed for the simulation of road car suspensions. The new family of models provides kinematic and elastic suspension models where the bushes can be simple linear models, nonlinear models or sophisticated elastomer models including frequency and amplitude dependent effects. Figure 6 shows the animation view of a rear multilink suspension in Dymola with two variants: the top image uses ideal joints and the lower one includes bushes.

The suspension models are defined using a template based approach with replaceable components allowing the links with ideal joints to be easily swapped for links with bushes at either ends. The bushes can then easily be redeclared to have the appropriate characteristics.

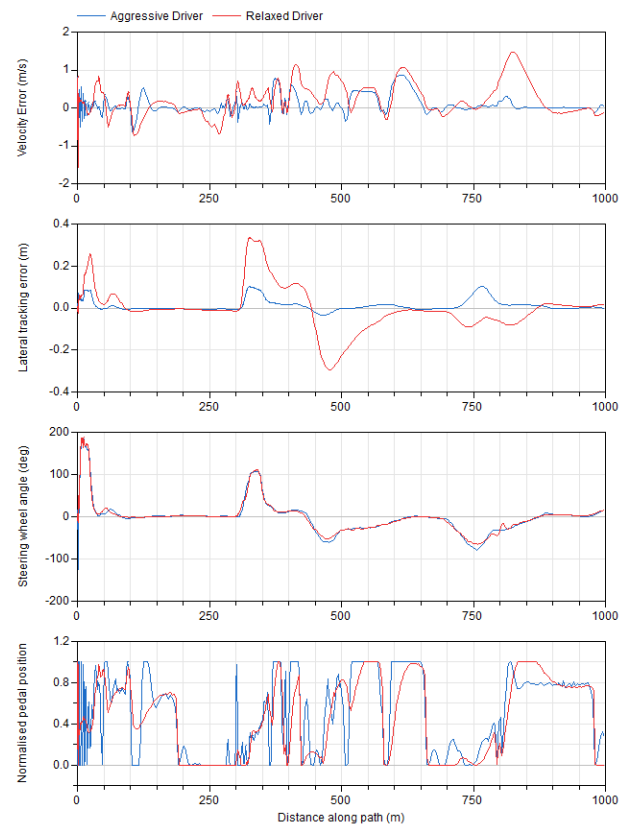


Figure 5: Comparison of aggressive (in blue) and relaxed (in red) driver model parametrizations. The plots show velocity tracking error, lateral tracking error, steering wheel angle and accelerator pedal position

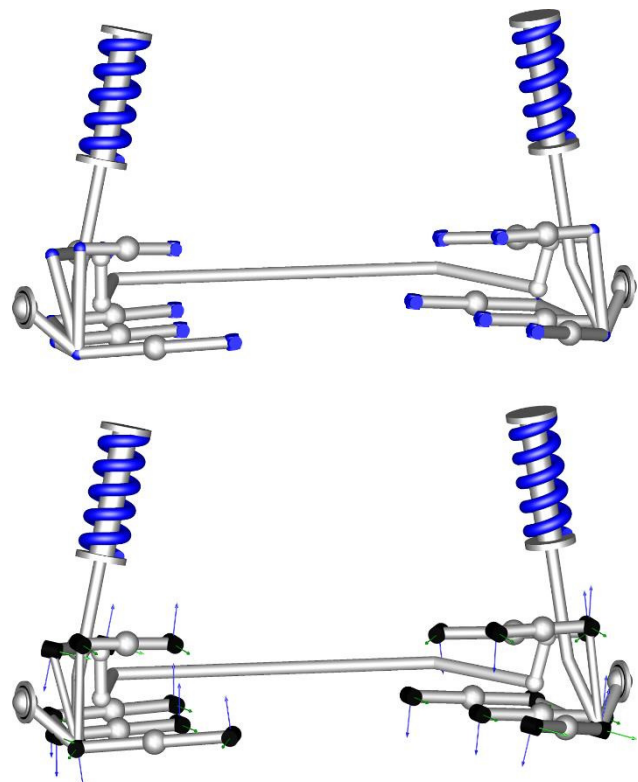


Figure 6: Kinematic Multilink suspension (top) and elastic Multilink suspension model (bottom)

The templates provide support for the easy integration of flexible bodies based on reduced Finite Element models to define the structural compliance of links, control arms, and uprights. Figure 7 shows an example of the quarter car template for a front McPherson strut suspension using bushes and a flexible body for the lower control arm. The upright can also be easily replaced with a flexible body as it defines separate attachment points for the wheel centre, damper strut, lower ball joint, track rod and anti-roll mechanism.

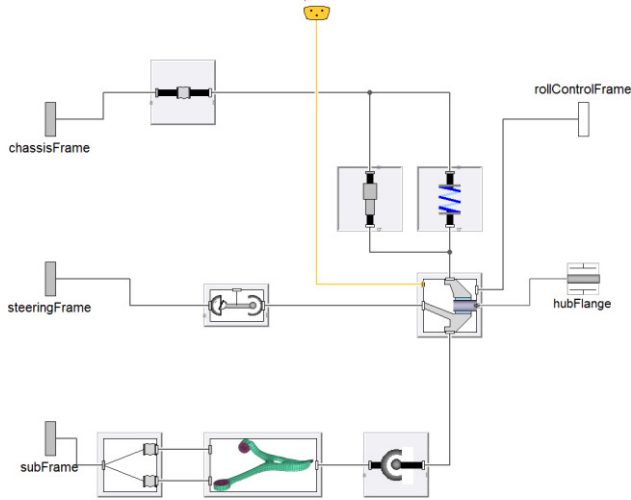


Figure 7: McPherson strut suspension with bushes and FE based lower control arm

4.2 Bush models

In road car applications the suspension bushes have a big effect on the ride and handling of the car, and have to be tuned to provide the right compromise between noise, vibration and harshness (NVH) and the desired handling characteristics. To support the tuning of these bushes at all stages of the design process a number of different characteristic models are provided including simple linear bushes and more sophisticated models with frequency and amplitude dependent characteristics.

In MultiBody simulation the modelling of elastomer mounts, such as suspension bushes, is usually done using a simple spring-damper element. This approach works satisfactorily provided the stiffness and damping terms are tuned to accurately capture the dynamic stiffness of the elastomer at the operating point being studied. In the case of superimposed oscillations with differing frequency and amplitudes, which an accurate road input would induce, then this approach is not adequate and more complex elastomer models have to be used (Persson, 2003).

A sophisticated elastomer model has been implemented that allows the frequency and amplitude dependency to be captured whilst maintaining a relatively low computation cost (Pfeffer, 2002) and an automatic calibration method simplifies the parametrisation of the model. Figure 8 shows how this

model is implemented in Modelica. It consists of a nonlinear force-displacement spring element that captures the static characteristic of the elastomer. In parallel with the spring element, is a linear damper and then a component array of frictional elements and spring-damper elements in series. The frictional elements are used to capture the amplitude dependent effects and the spring-damper in series, often referred to as a Maxwell model, are used to capture the frequency dependent effects.

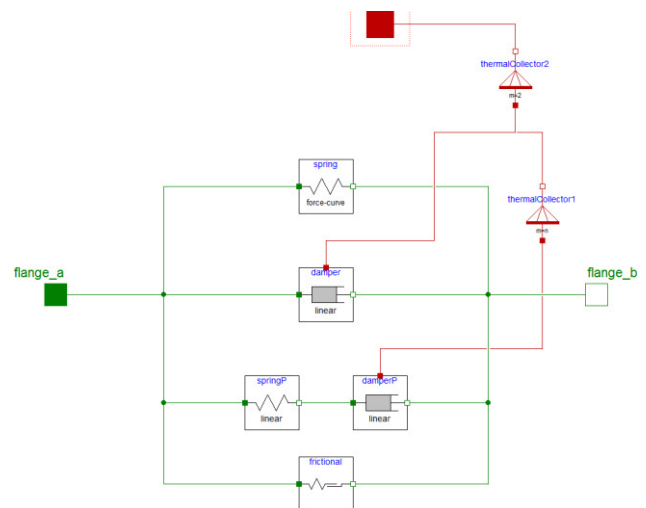


Figure 8: Elastomer with frequency and amplitude dependent characteristics

To parameterise the model the elastomer needs to be measured to get the static and dynamic stiffness and loss angle characteristics. Using this information, optimisation can be used to tune the model parameters, provided the user first decides on the sizes for the component arrays. The number of Maxwell models included determines the ability of the model to cover the frequency range of interest, with 1 Maxwell model it is possible to accurately represent the bush at one of the measurement points but at other frequencies the dynamic stiffness and loss angle will be incorrect, as shown in Figure 9. In the example shown, the parameters for the Maxwell model were calculated from the measurement data at 21Hz using the following method. The measurement results for dynamic stiffness and loss angle can be used to calculate the complex stiffness of the bush:

$$k_{dyn} = k_d(\cos \alpha + j \sin \alpha) \quad (1)$$

Where k_d is the dynamic stiffness and α is the loss angle. The complex stiffness of the bush, without frictional elements, at a specific frequency can be calculated according to:

$$k_{dyn} = k_e + c \cdot w \cdot j + k_{maxwell} \quad (2)$$

Where k_e is the elastic stiffness, c is the damping, w is the frequency in rad/s, and $k_{maxwell}$ is the complex stiffness of the Maxwell model. From equations 1 and 2 we can calculate the stiffness contribution that must

come from the Maxwell model. The complex stiffness of the Maxwell model is given by:

$$k_{maxwell} = k_m \left(\frac{w \cdot j}{1 + t_r \cdot w \cdot j} \right) t_r \quad (3)$$

Where k_m is the stiffness of the spring and t_r is the time response of the Maxwell model. $k_m \cdot t_r$ is the damping of the Maxwell model. Equations 1-3 can be solved as a nonlinear system consisting of 2 unknowns k_m and t_r to calculate the properties of a single Maxwell model tuned to work correctly at the selected frequency.

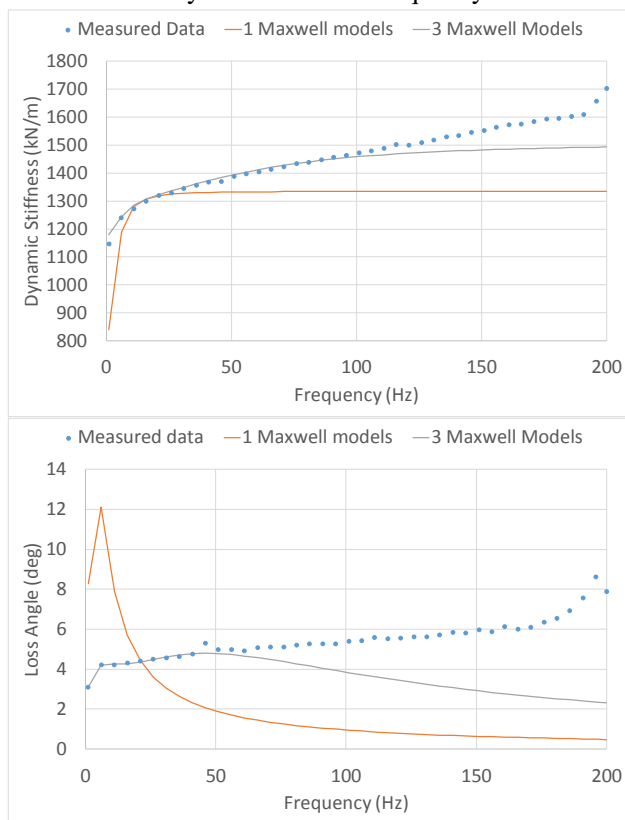


Figure 9: Dynamic stiffness and loss angle for a tuned model with varying numbers of Maxwell models

By increasing the number of Maxwell models in parallel to 3 and through the use of optimization to determine the parameter values it is possible to get the bush model to have a greater accuracy over a larger frequency range. Figure 9 shows the results of tuning the parameters to give a good fit between 1 and 46Hz. This would mean that the bush works well in the frequency ranges necessary to simulate primary and secondary ride effects. To cover a wider frequency range then additional Maxwell models would be needed but this increases the difficulty of the optimization problem, and as each Maxwell model adds 1 state to the model it also increases the computation time.

5 Real-time simulation

One of the key advantages of a Modelica based approach for vehicle dynamics analysis is the ability of Dymola to export simulation code that is capable of

running in real-time. This has been used in Motorsport for running vehicle models as part of a Driver-in-the-loop simulator for several years (Dempsey, 2012). Typically though a race car does not include bushes in the suspension which makes the vehicle dynamics model simpler and easier to run in real-time even with structural compliance effects included. To run a suitably detailed road car vehicle dynamics model in real-time we need to take advantage of new capabilities in Dymola 2016 to parallelize a model (Elmqvist, 2014; Andreasson 2014).

Utilizing this approach we have been able to partition the model into a number of separate computation tasks: the body; front suspension; the left and right rear suspensions; and 4 tyre models. The model used is a saloon car with double wishbone front suspension and a multilink rear suspension. The front suspension uses ideal joints but includes compliance effects in the upright and the rear suspension includes bushes at the inboard and outboard ends of every link, see Figure 10. The powertrain model includes a mapped engine model, a 6 speed automatic gearbox with torque converter and it is front wheel drive. The model has been optimized to eliminate events and uses elastic friction models for the brakes and losses within the powertrain. The whole vehicle model consist of 243 states, of which 17 relate to the brakes and powertrain systems and the remainder are related to the suspension and tyres.

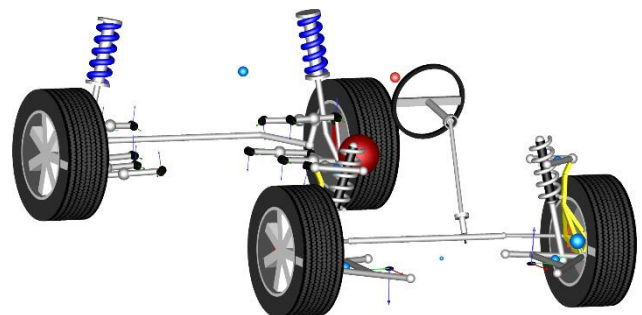


Figure 10: Animation view of the vehicle model

The model is partitioned into separate tasks using the decouple blocks available in Dymola as shown in Figure 11 for the rear suspension. The decouple blocks are used to break down the size of the implicit nonlinear system of equations related to the inline integration method. In the full vehicle model, if the decouple blocks are not used there is a large implicit nonlinear system of equations of size 139 but when these blocks are used this is broken up in to a set of 4 nonlinear systems of equations of sizes 21, 6, 38, and 38. The two systems of size 38 are related to the left and right rear suspensions. These smaller systems of equations are easier to calculate, and most importantly the jacobian used by the implicit inline integration method is much smaller and easier to compute.

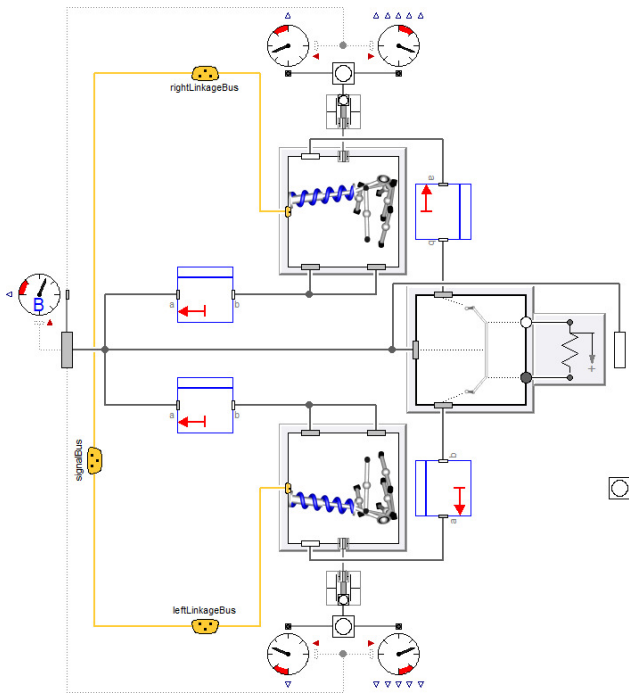


Figure 11: Rear suspension model including decouple blocks to partition the model into separate tasks for the left and right suspension

By splitting the model into these subtasks we have been able to run the model in real-time using an inline implicit Runge Kutta solver with a 1ms time step. Figure 12 shows the turnaround time for the model during a few laps of the Motorland Circuit of Aragon when running the vehicle with rFpro on a PC workstation. This track model uses high fidelity LiDAR data to define the surface which is provided to the model by rFpro TerrainServer as described in 2.1. The PC runs Windows 7 (64 bit) with an Intel Core i7 5960X processor overclocked to 4.2 GHz. There are very occasional overruns but these are small in magnitude, around 0.2ms, and infrequent which means they can be easily tolerated by the DiL system.

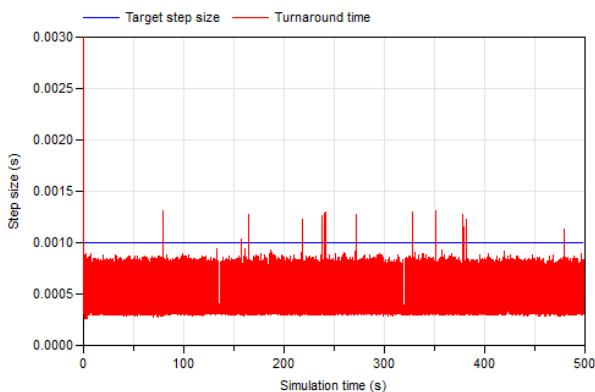


Figure 12: Turnaround time for the vehicle model running in rFpro on a track using LiDAR data

When configuring a model for use in real-time simulation there is always a trade-off to be made

between performance and accuracy compared to running the model with a variable step solver. It is important to verify that the change in simulation results are minor when partitioning the models and to find the best compromise between computation time and accuracy. Figure 13 shows a comparison of the simulation results obtained with this model driving at 55kph with a sinusoidal steering input. It shows that the variations found with the different real-time solver settings are small when comparing the results to those achieved with a variable step solver.

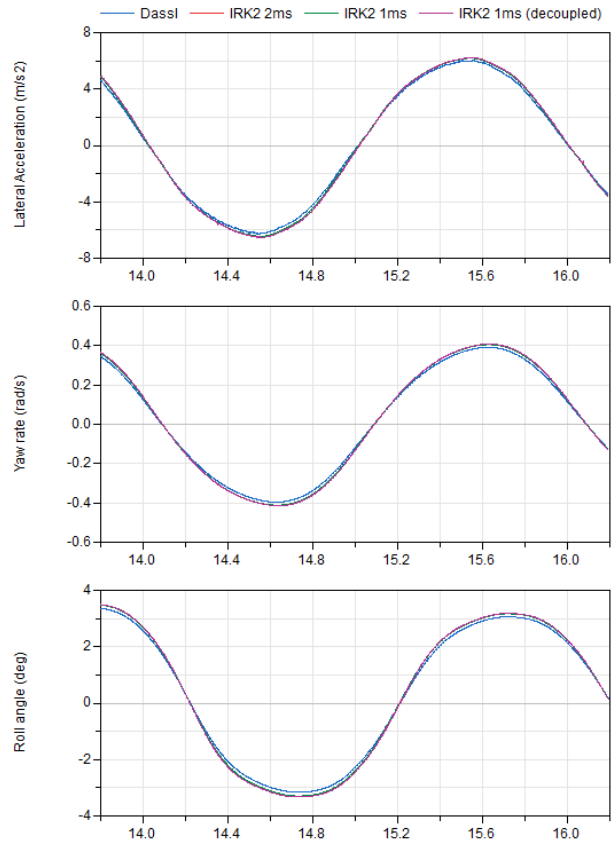


Figure 13: Comparison of lateral acceleration (top), yaw rate (middle) and roll angle (bottom) for the model running with Dassl compared to using Implicit Runge Kutta 2nd order fixed step solver at rates of 1ms, 2ms and with the decoupled model running at 1ms

6 Conclusions

The integration between the driver-in-the-loop system, rFpro and Modelica has been extended to enable the same track data to be used for offline and real-time simulation. This means the analysis work in Dymola can use the same high fidelity track data that is available to the driver-in-the-loop simulator. A new closed loop driver model has also been developed for use with these high fidelity tracks and it allows different driver behaviour to be assessed as well as comparing the handling effects of vehicle setup changes. These capabilities are available in a commercial Modelica library called TerrainServer.

New suspension models have been developed to support the transfer of these technologies from motorsport into road car applications. These new models enable higher fidelity suspension models to be created, and when coupled with the latest enhancements in Dymola, support parallelisation across multiple cores enabling a full MultiBody vehicle model with bushes in the suspension to be run in real-time at 1 kHz using standard PC hardware. These models will be available in future commercial Modelica libraries.

References

- J.Andreasson, H. Elmqvist, J. Griffin, D. Henriksson, S.E. Mattson, H. OlBon, Real-Time Simulation of Detailed Vehicle Models using Multiple Cores, *12th International Symposium on Advanced Vehicle Control*, September 22-26, 2014, Tokyo, Japan
- M. Dempsey, A.Picarelli, G.Fish, Using Modelica models for Driver-in-the-loop simulators, *Proceedings of the 9th International MODELICA Conference*, September 3-5, 2012, Munich, Germany, doi: 10.3384/ecp12076571.
- H. Elmqvist, S.E. Mattson, H. OlBon, Parallel Model Execution on Many Cores, *Proceedings of the 10th International Modelica Conference*, March 10-12, 2014, Lund, Sweden, doi:10.3384/ecp14096363
- C. Hoyle, How Mobile Lidar is Revolutionising Automotive Testing, *LiDAR News Magazine*, Vol. 4 No. 7, Spatial Media, 2014
- H. Pacejka, *Tire and Vehicle Dynamics* (Third edition), *Butterworth-Heinemann*, 2012, doi:10.1016/B978-0-08-097016-5.01001-9
- A. Persson and F. Karlsson, *Modelling Non-Linear Dynamics of Rubber Busings – Parameter Identification and Validation*, Division of Structural Mechanics, Lund University, 2003, <http://lup.lub.lu.se/student-papers/record/3566697>
- P. Pfeffer and K. Hofer, Simple Non-Linear Model for Elastomer and Hydro-Mountings, *ATZ worldwide* 5/2002 Volume 104, 2002
- rFactor Pro, *HD Terrain Server User Guide*, September 2014
- Toso, A. and Moroni, A., Professional Driving Simulator to Design First-Time-Right Race Cars, *SAE Technical Paper* 2014-01-0099, 2014, doi:10.4271/2014-01-0099.

Modeling and Validation of a Multiple Evaporator Refrigeration Cycle for Electric Vehicles

Andreas Varchmin¹ Manuel Gräber² Jürgen Köhler¹

¹Technische Universität Braunschweig, Institut für Thermodynamik (*a.varchmin@tu-bs.de*)

²TLK-Thermo GmbH

Abstract

Multiple evaporator vapor compression cycles become relevant for thermal systems in electric vehicles since batteries and other electric components demand cooling for a secure operation. In difference to most other applications with parallel evaporators cooling demands and temperature levels vary between the different secondary fluids. This leads to a more complex system behavior that needs to be described for optimality and control analysis. In this paper a dynamic model for an automotive air conditioning cycle with an additional evaporator for battery cooling is developed and validated. A battery model library for calculating temperatures and waste heat flows of battery cells and modules is presented. Multi-evaporator effects and their consequences are discussed. Reasonable actuating and control variables are chosen and a discussion regarding possible control schemes is given.

Keywords: Multi-Evaporator Cycle, Parallel Evaporators, Vapor Compression Cycle, Electric Vehicle, Relative Gain Array

1 Introduction

Battery electric vehicles (BEV) or plug-in hybrid electric vehicles (PHEV) require a more complex thermal system and vapor compression cycle than conventional vehicles. In many cases heat needs to be gained, respectively led away, at more than one spot. One important spot is the battery that needs to be in a reasonable and safe temperature zone. Depending on the ambient conditions the heat gathered at the battery or the power electronics can be used for operating a heat pump.

For maximum lifetime battery temperatures should lie between 15°C and 35°C and show small temperature gradients over battery cells and modules (Pesaran et al., 2013). Maintaining these temperatures leads to the necessity of active battery cooling. The most common way of active cooling are cooling plates under or between battery cells. These plates are flown through with a cooling liquid like a water-glycol-mixture but can also be used to

directly evaporate a refrigerant. Based on a typical air-conditioning unit that owns an evaporator for air cooling a second evaporator is needed to cool battery and possibly power electronics, too.

A heat pump might be necessary for heating the cabin at low temperatures. In conventional vehicles heat from the engine's exhaust is used for heating. In BEVs the battery is the only source of energy which is, due to energy densities of current batteries, small compared to conventional vehicles. A regular electric heater would heavily shorten the vehicle's range. Therefore switchable vapor compression cycles that can function as air conditioning or heat pump have been designed (Ahn et al., 2015). When operating as heat pump the cycle might gain heat from the ambient but also, if available, from battery, electric motor or power electronics. In this case a second evaporator is also necessary.

Multiple evaporator vapor compression cycles are known in different applications. In supermarket refrigeration cycles a high number of evaporators are connected in parallel so that every cabinet can be cooled independently. Mostly thermostatic expansion valves are used to regulate the fluid flow in every cabinet. Furthermore there are systems with more than one compressor unit to create two pressure levels for refrigerators and freeze cabinets. These two pressure levels are connected in parallel as well (Titze et al., 2013). Another multi-evaporator application are HVAC systems in buildings. Especially in commercial buildings so called variable refrigerant flow systems are increasingly used with multiple evaporators for locally distributed cooling zones (Elliott et al., 2011). Usually parallel evaporators are used in these applications. Vapor Compression Cycles with serial connected evaporators exist in fewer applications, e.g. in simple fridge freezers.

In contrast to the described systems the operation of vapor compression cycles in vehicles is a lot more transient. Furthermore the temperatures of the different secondary fluids can be apart whereas temperatures in supermarket cabinets, respectively cooling zones in buildings, share a nearly identical temperature. Since the thermal system needs to be as compact as possible, only one pressure level for both evaporators is practicable (see fig-

ure 1). This makes the optimal operating point less evident and leads to advanced control schemes. Modeling approaches for automotive air conditioning cycles in Modelica with particular attention to evaporators have been published by Limperich et al. (Limperich et al., 2005).

For a better understanding and for fast controller design a simulation model for an automotive multi-evaporator system is developed in this paper. It is concentrated on battery and cabin cooling instead of heat pump operation in electric vehicles. Measurement data obtained from a test bench is used for validation of the model and the system is analyzed for a suitable controller design. Fundamental effects and interdependencies of the evaporators are shown. Gained knowledge is then used to control the multi-evaporator system under conditions of a transient real-life driving cycle.

Structure of the Paper

In section 2 a model of an automotive air conditioning system with parallel evaporators for cabin and battery cooling is described. In section 3 follows a description of an electric-thermal model of a lithium-ion battery for calculating temperatures and heat flows of cells and modules. An overview of a test bench containing three plate heat exchangers is given in section 4. All secondary fluids in this system are liquids since measurement data can be obtained easily and fast for this configuration. Section 5 shows the interaction of actuating and control variables as superheats or cooling capacities. Moreover the parallel evaporator model is validated with measurement data. Section 6 analyzes the interdependencies of actuating and control variables. With the help of relative gain arrays (RGA) possible controller schemes are discussed. In section 7 simulation results for the parallel evaporator system are shown. Section 8 summarizes the paper and proposes possibilities for future work.

2 Automotive Air Conditioning System with Additional Evaporator for Battery Cooling

In the following a dynamic model of a parallel evaporator vapor compression cycle for electric vehicles including battery cooling is described, see figure 1. A description of the implied control scheme is given in section 6. The system has two parallel paths on the low pressure side and consists of a compressor, three heat exchangers, two electronic expansion valves (EXV) and a receiver. As refrigerant R134a is used. The compressor is a variable speed scroll compressor. The receiver is located on the high pressure level behind the condenser so that there is zero subcooling in mostly all operating points since the receiver separates gas from the liquid. This does not lead

to the most efficient behavior because small degrees of subcooling normally have a positive impact on the coefficient of performance (COP). The condenser is a multi port extruded tube (MPET) heat exchanger, situated at the vehicle's front end. Ambient air is used as the condenser's secondary fluid, leading the heat out of the cycle. In modern cars often condensers with an integrated receiver and an additional subcooling zone are used for optimal COPs. Behind the receiver the refrigerant path is divided and leads to an EXV each. In the first path there is located one MPET heat exchanger for cooling the cabin air. In the parallel path a plate heat exchanger is used for cooling the battery. Due to evaporating temperatures falling occasionally under 0°C a 50/50 mixture of water and glycol flows through the secondary path of this evaporator. Nearly all model approaches are based on the model library *TIL* (see Richter (2008)). Physical properties are calculated with *TILMedia* (see Schulze (2013)).

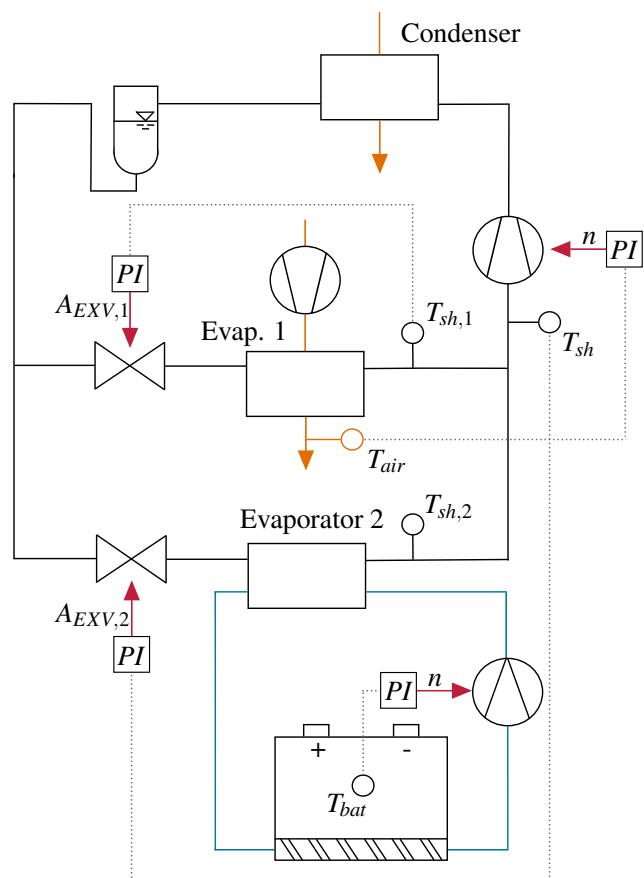


Figure 1. Vapor Compression Cycle for a Battery Electric Vehicle including Battery Cooling

The compressor is modelled dynamically with a loss-based approach, including leakage and friction losses (Schedel et al., 2013). The model is fitted to the compressor found in the test bench described in section 5. The EXV model is based on Bernoulli's principle and is therefore a static model. For modelling the heat exchangers a finite volume approach with a variable number of

volumes is used. One volume is described by three cells for refrigerant, liquid and wall material. The liquid and air cells contain transient balances for mass and energy, the refrigerant cell additionally owns a momentum balance. The pressure time derivative is constant over all refrigerant cells of one pressure level. This can be interpreted as a neglect of sonic effects which is appropriate for systems that are not characterized by fast dynamics (Gräber et al., 2010). The receiver model owns one volume with a transient mass and energy balance and has a varying filling level.

The valve openings $A_{EXV,1}$, $A_{EXV,2}$ and the compressor speed n are actuating variables of the primary (refrigeration) cycle. Typical control variables would be the two superheats $T_{sh,i}$, the first evaporator's air outflow temperature T_{air} and the battery temperature T_{bat} . Alternatively the cooling capacities \dot{Q}_i of both evaporators could be used. Referring to section 6 it becomes apparent that in difference to a standard refrigeration cycle the numbers of actuating variables and control variables of the primary cycle are not equal.

3 Battery Model

To integrally describe thermal systems of electric vehicles as the one discussed in this paper, the behavior of the battery needs to be represented. Important approaches and equations are published in the Modelica Energy Storage Library (Einhorn et al., 2011). For a good suitability and usability regarding automotive systems an electric-thermal battery model library has been developed. The library is designed as an add-on to the component library *TIL* and provides models for battery cells, modules and systems. It is focused on describing temperatures, waste heats and cooling approaches.

In the library battery systems can be discretized down to cell level. Cells can be connected in series or parallel. Each discretization element owns a state of charge (SOC), an electric equivalent circuit, a dynamic energy balance and calculates arising heat flows. The state of charge is coupled with the open circuit voltage. The equivalent circuit can be switched between a model with a single ohmic resistance and an impedance based model. Heat results from different processes inside the battery cells. Losses can be divided in irreversible and reversible heat flows:

$$\dot{Q}_{irr} = -I^2 R_i \quad (1)$$

$$\dot{Q}_{rev} = T \Delta S \frac{I}{nF} \quad (2)$$

The irreversible heat is dependent from the electric current, that is defined by the power demand of the motor and from the impedance which itself is dependent from cell temperature and SOC. The reversible heat is dependent from entropy differences that occur at the electrodes, e.g. because of lithium intercalations in lithium-

ion cells. The equation is only exact for isothermal processes but may also be used when temperature changes. ΔS is the molar entropy change while n represents the number of electrons per reaction and F the Faraday constant. These differences are cell dependent and are functions of the SOC. This approach is described extensively by Viswanathan et al. (Viswanathan et al., 2009). They have also published data for various cell types in the same paper.

The model used in the automotive system described in the previous section is adapted to a battery system in a BEV with 300 prismatic lithium-ion cells. The modules are chilled by cooling plates underneath and contain twelve cells each, at which each three cells are connected in parallel. Water-glycol is used as cooling liquid.

4 Test Bench

The test bench contains a refrigeration cycle similar to the one described in section 2. It is dimensioned to fit an automotive air conditioning and cooling application. One difference is that all three heat exchangers are plate heat exchangers receiving and giving off heat by secondary loop cycles filled with water (condenser) or respectively water-glycol mixture (evaporators) as shown in figure 2. This originates from easier and more accurate measuring when handling liquids instead of gases. Both evaporators have same dimension and are of the same type. Another difference to the system model is that a battery is not included since it is focused on the multiple evaporator behavior. Anyway the inflow liquid temperatures of the evaporators $T_{liq,1}$ and $T_{liq,2}$ can be manipulated by electric heaters. Furthermore the mass flow rates $\dot{m}_{liq,1}$ and $\dot{m}_{liq,2}$ are controlled as well.

The scroll compressor works with direct current and variable speed. The EXV is driven by a stepper motor with constant adjusting speed. At various positions temperatures and pressures are measured. The refrigerant mass flow rate is measured behind the compressor in the pure gas zone. The primary cycle of the test bench is pictured in figure 3.

All temperatures are measured with thermocouples. The liquid mass flow rates are measured with magnetic flow meters while the refrigerant mass flow rate behind the compressor is measured by coriolis principle. This means that the cooling capacity measuring of the evaporators are based on the liquid side when both evaporators are in use. The quality of stationary measurements can be controlled with the help of results that are gathered by only one evaporator operating. In this case one cooling capacity can be obtained based on the refrigerant side using the more accurate coriolis flow meter and compared to the measurements based on the liquid side.

very well with a maximum deviation of 160 W. Results for capacity 1 vary up to 365 W.

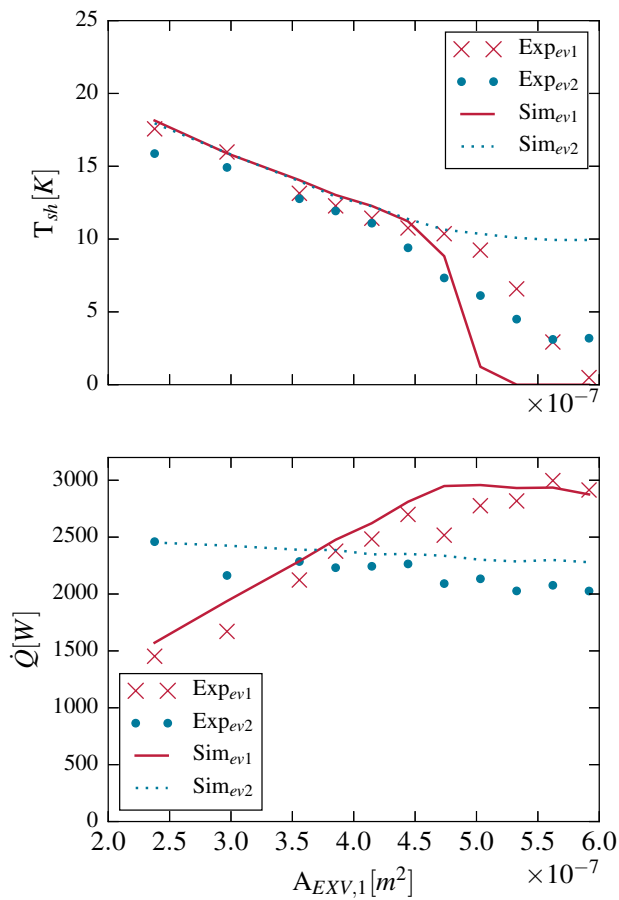


Figure 5. Superheats and Cooling Capacities of both Evaporators Depending on Valve Opening $A_{EV,1}$ with Constant Valve Opening $A_{EV,2} = 3.7 \cdot 10^{-7} m^2$

Figure 5 also shows superheats and cooling capacities. The compressor speed and the valve opening area $A_{EV,2} = 3.7 \cdot 10^{-7} m^2$ are set constant but $A_{EV,1}$ is varied. At low opening areas the superheats again reach a maximum value of the superheat. Anyway both superheats drop because of a rising low pressure. At $A_{EV,1} = 4.5 \cdot 10^{-7} m^2$ the simulated superheat 1 starts to fall faster, the measured superheat 1 starts falling faster at $5.0 \cdot 10^{-7} m^2$. Both reach zero superheat with nearly the same opening difference. The measured values of superheat 2 fall as well (to 3 K) while the simulations state that values still reach maximum superheat. The first evaporator's cooling capacity rises with higher openings of its corresponding valve. The second evaporator's capacity falls slightly since mass flow rates through the second path drop. Apart from one measurement point at $A_{EV,1} = 4.7 \cdot 10^{-7} m^2$ which may be due to a non-stationary operation while measuring, simulation results of the capacities fit to measurement data very well. Summarizing it can be stated that a valve opening area influences the evaporator in its own path more than the other

one and that superheats share the same trend while cooling capacities go in different directions.

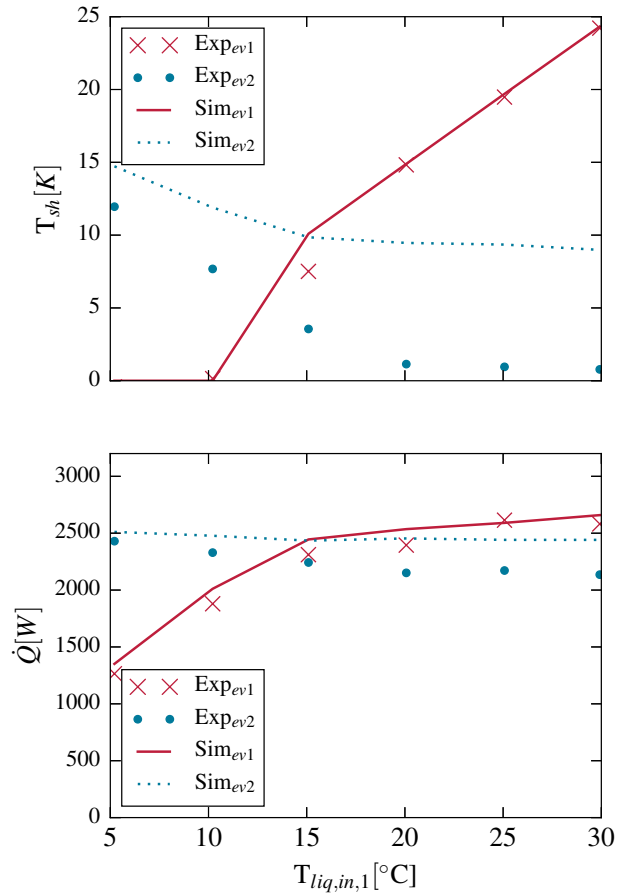


Figure 6. Changing of Superheats and Cooling Capacities with Varying Liquid inflow Temperature $T_{liq,in,1}$

The investigation shown in figure 6 deals with a changing boundary condition, precisely the secondary fluid inflow temperature of evaporator 1. The temperature changes from $5^{\circ}C$ to $30^{\circ}C$ while the temperature of the second liquid stays constant at $15^{\circ}C$. Deviant secondary fluid temperatures of more than a few Kelvins would not be typical for building or supermarket applications but occur in vehicles, e.g. if the cabin is already cooled down while battery and power electronics produce a lot of waste energy. In fact in many cases it will be more efficient to operate the battery at temperatures up to $35^{\circ}C$, needing smaller cooling capacities, while the cabin temperature is controlled to around $20^{\circ}C$. It can be seen that superheat 1 rises with rising temperature as one could have expected. Superheat 2 drops because the low pressure level is lifted with higher evaporating temperatures. The same behaviors are observed for cooling capacities. The evaporator with higher secondary fluid temperature transports more heat. The overall heat capacity reaches a maximum at same secondary fluid temperatures, meaning also a maximum efficiency. Simulation results of evaporator 1 fit very well to measurement data while the reaction to deviant

temperatures is not optimal at evaporator 2. It needs to be considered that small deviations of the exchanged heat lead to high deviations in superheat. Only a small part of the heat is used for superheating while most of it vaporates the refrigerant.

Some of the occurring main effects can be summarized:

- The Compressor speed influences both evaporators but has more effect on the evaporator with higher cooling capacity. It nearly has no effect on the lower cooling capacity.
- Changes in EXV opening areas lead both superheats in the same direction while cooling capacities move in opposite directions. EXVs has a higher influence on the evaporator in its corresponding path.
- Different secondary fluid temperatures lead to different cooling capacities. Control variables shift to opposite directions when one temperature is changed.

With regard to the validation results it can be stated that the occurring effects are represented in the parallel evaporator model, also the ones that may not be clearly self-explanatory. In further work more investigations have been done with deviant actuator variables and boundary conditions that also lead to interesting effects. Deviations between simulation and experiment data are similar to the ones shown here.

6 System Analysis

The previous sections shows effects of a multi-evaporator system that need to be analyzed systematically. With the help of the validated model the interdependencies can be quantified.

A good overview of design and operation of a vapor compression cycle with a single evaporator is given by Jensen and Skogestad (2007). They state that a simple vapor compression cycle is optimally designed if there is no superheating and a small degree of subcooling. To reach zero superheating, which means an optimal heat transfer in the evaporator, a low-pressure receiver can be used in simple cycles. In a multiple evaporator cycle a low-pressure receiver would ensure a secure operation of the compressor but not lead automatically to an optimal operation point. An overall superheat of zero does not mean zero superheat in each of the evaporators. It will be more likely that the outflow refrigerant of one evaporator has a high degree of superheat while the other evaporator's outflow state lies inside the two-phase region. Both superheats need to be controlled and the advantage of a low-pressure receiver disappears. In automotive air conditioning systems reaching a small degree of subcooling is often done by a condenser unit with integrated receiver and a short subcooling zone. This paper concentrates on

evaporators so that for an easier test bench setup there is no subcooling zone behind the high-pressure tank (see figure 2).

From an operation view the compressor work (here speed) and the valve openings can be manipulated. The active refrigerant mass is adjusted by the receiver. Since both superheats need to be controlled and it is desirable to control both cooling capacities (or the outflow temperatures $T_{liq,1}$) one actuating variable is missing to regulate all four control variables independently. In many multi-evaporator systems, e.g. in the ones mentioned in the introduction, the cooling capacities do not need to be controlled separately because the temperature setpoints of the different secondary fluids are equal. In the case of similar thermal load the control variables can be controlled together.

One obvious way to control the cooling capacities separately is manipulating a secondary fluid mass flow rate \dot{m}_{liq} (e.g. through pump speed). The mass flow changes the effective heat transfer in the corresponding evaporator. This leads to an optimization problem between pump work and primary cycle's efficiency. Moreover the mass flow rates are often given by other conditions. In the system analysed here manipulating the battery cooling liquid mass flow rate seems to make sense.

The described interdependencies between the control variables demand special controller designs. Reasonable approaches could be a decoupling control that prevents change in other process variables when one variable is changed or a dynamic feedforward control that has knowledge about the physical behavior as described in previous work (Varchmin et al., 2014). However development of advanced control schemes is not the main focus of this paper. Hence, Single-Input-Single-Output (SISO) controllers are used for the automotive system described in section 2, even if a decoupling seems to be recommended as it is shown in the following.

$n = 3000rpm$	n	$A_{EXV,1}$	$A_{EXV,2}$	$\dot{m}_{liq,2}$
\dot{Q}_1	0.38	0.56	0.05	0.01
\dot{Q}_2	0.33	0.04	0.61	0.02
T_{sh}	0.05	0.03	0.4	0.52
$T_{sh,1}$	0.24	0.36	-0.06	0.45

$n = 6000rpm$	n	$A_{EXV,1}$	$A_{EXV,2}$	$\dot{m}_{liq,2}$
\dot{Q}_1	0.66	0.14	0.07	0.12
\dot{Q}_2	0.33	0.02	0.32	0.32
T_{sh}	-0.12	-0.06	0.81	0.38
$T_{sh,1}$	0.13	0.9	-0.2	0.17

Table 1. Relative Gain Arrays of Two Evaporator System at Different Compressor Speeds

For Multi-Input-Multi-Output (MIMO) systems steady-state interactions, e.g. the ones shown in the previous section, can be represented in relative gain arrays (RGA). These matrices show normalized steady-state gain information and are therefore a measure of interactions (Grosdidier et al., 1985). In table 1 two RGAs derived by the validated model are shown. RGAs quantify system behavior and may be used to develop SISO (as well as MIMO) controller schemes, i.e. to obtain reasonable pairs of actuating and control variables (Skogestad and Postlethwaite, 2007). The arrays are different for every operation point, only for a linear system one array would be valid globally. Gain scheduling could be used to adapt controller parameters to nonlinear behavior in dependence of the current operating point.

The displayed arrays are valid for two different compressor speeds, i.e. cooling capacities, at small superheats. Instead of the superheat of evaporator 2 the overall superheat in front of the compressor is chosen as a control variable. Firstly the overall superheat is most relevant for a secure compressor operation and secondly the RGAs are less coupled in this case. Optimal actuator-control pairs for the operation points are printed in bold. It can be seen that the system is highly nonlinear and that actuating variables that are a good choice in one point may be a useless choice in another. While the reactions to changes of compressor speed n stay similar (first column) the reactions of the other actuating variables differ a lot. E.g. the mass flow rate \dot{m}_{liq} would be best paired with cooling capacity \dot{Q}_1 for $n = 6000rpm$ (value 0.32) but has nearly no effect for $n = 3000rpm$ (0.02).

The RGAs do not give any information about the system dynamics. Similar time constants complicate SISO control and may lead to instabilities. This may be due to two controllers having influence on the same control variables that wind up to not desired behavior because they disturb each other simultaneously. This problem will be discussed in combination with advanced controller designs in future work.

7 Simulation Results

System analysis leads to a control scheme for the automotive system that is shown in figure 1. Four SISO Controllers implemented as PI controllers are used for reaching reasonable setpoints. As discussed table 1 shows possible pairs of actuating and control variables. Values change for different compressor speeds and so a compromise between the operation points needs to be found. Speed n and opening area $A_{EXV,1}$ can be chosen without issues for \dot{Q}_1 and $T_{sh,1}$ because of high values in both matrices. It seems best to choose opening area $A_{EXV,2}$ for \dot{Q}_2 and the liquid mass flow rate $\dot{m}_{liq,2}$ for the overall superheat T_{sh} because of satisfying values for these pairs in both matrices. A problem for this configuration is that

in some operating points the battery would not be able to provide enough heat to ensure a sufficient superheat. The secondary liquid cycle would need to be designed for very high heat flows that do not occur very often. Therefore the controller scheme of the lower matrix in table 1 is chosen although the dependency between $\dot{m}_{liq,2}$ and \dot{Q}_2 is not very strong for slow compressor speeds. It may be a benefit that at high battery temperatures the compressor speed will probably lie over 3000 rpm so that there would be a sufficient coupling again.

The model is simulated with a transient real-life driving cycle (leading from Braunschweig to Wolfsburg) as a boundary condition (see figure 7). The ambient temperature is 30°C and all components possess this temperature at the start of the simulation. The air flowing into the cabin has a temperature of 10°C and the setpoint for the superheats is 7K. For the battery temperature controller a setpoint of 30°C is chosen whereat it is considered that this point will not be reached at all times. The inner battery temperature has a high time constant since the battery owns a great heat capacity and the heat needs to be conducted through the cell until it is led away by the liquid cycle. Only by predictive control it could be controlled perfectly.

Results are shown in figure 7 (b)-(f). At the beginning of the cycle the compressor speed is high to cool down the air and evaporator masses (c). After about one minute the compressor slows down to approximately 3000 rpm. The temperature of the inflowing air reaches its setpoint of 10°C quite fast and remains at this level for the whole cycle (d). The battery temperature rises slowly until around 800 s (f). From that time on the driving cycle reaches higher velocities and drive power rises. This leads to a faster rising battery temperature. Hence, the cooling liquid mass flow rate is set higher (b). This shows effect at about 800 s so that the battery temperature slope is eased and in the following negative. As discussed previously the main target of battery cooling is not reaching a certain setpoint but limiting the maximum temperature to 35°C. Superheats stay in a secure range except for a small peak at 50 seconds. The overall superheat rises between 200 and 1000 seconds (e). The EXV in the battery cooling path reacts to this deviation (not displayed) but can not balance perfectly since the mass flow rate of the battery cooling liquid is also rising until 1000 seconds.

8 Conclusion

In this paper a dynamic model for an air conditioning cycle with an extra evaporator for battery cooling has been developed. A library with physical models for battery cells and modules has been described. Multi-evaporator effects were shown and explained. The developed cycle model has been validated with the help of a test bench containing plate heat exchangers. Elaborated interdependen-

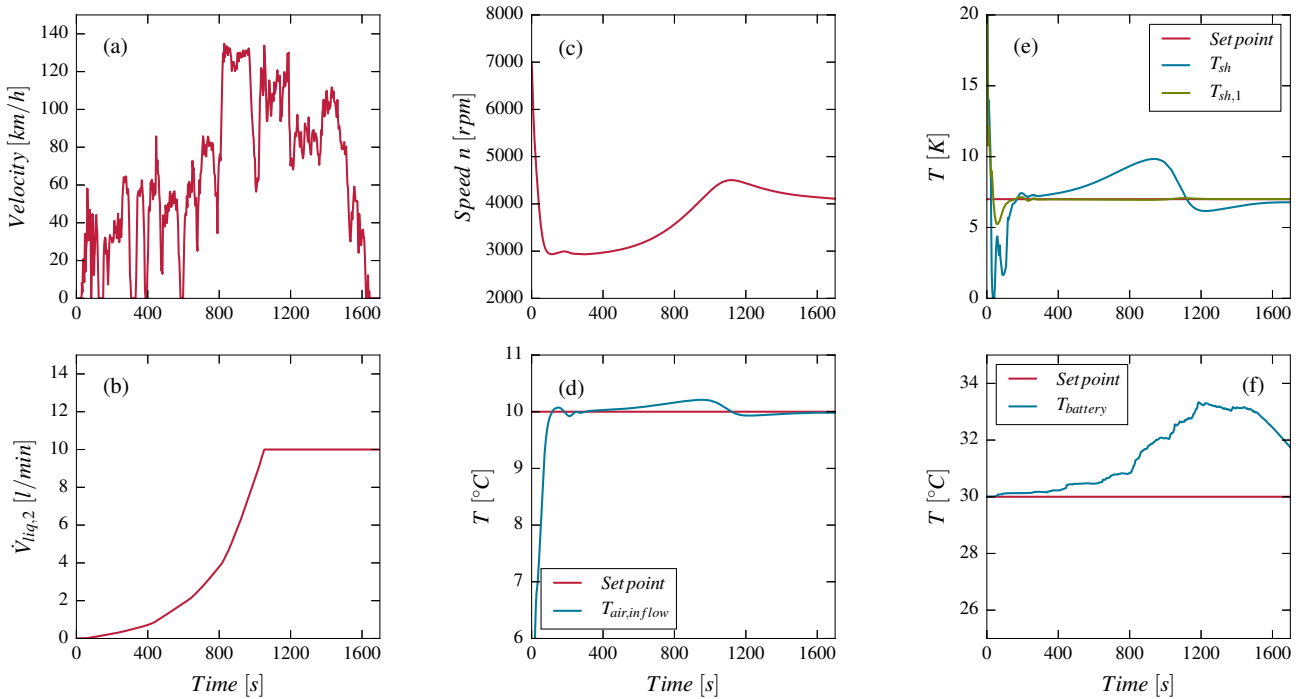


Figure 7. Simulation Results of the Automotive System due to a Driving Cycle

dencies were exemplarily quantified by relative gain arrays. One of the results is that in difference to a standard vapor compression cycle only three actuating variables of the primary cycle exist for four control variables. This leads to an optimization problem instead of simple optimality conditions. Based on this analysis a SISO controller scheme was derived and simulation results were presented. With the developed model different battery types and cooling approaches can be tested. Furthermore advanced control schemes can be tested and a system efficiency analysis can be performed.

Future work will deal with analysis of multi-evaporator system dynamics and the arising optimization problem. Moreover advanced multivariable controller designs could be developed for an optimal and secure operation.

Acknowledgements

This work has been supported by the German Ministry BMBF in the *Reflex Thermo* project.

References

- J.H. Ahn, H. Kang, H.S. Lee, and Y. Kim. Performance characteristics of a dual-evaporator heat pump system for effective dehumidifying and heating of a cabin in electric vehicles. *Applied Energy*, 146:29–37, 2015. doi:10.1016/j.apenergy.2015.01.124.
- M. Einhorn, F.V. Conte, C. Kral, C. Niklas, H. Popp, and J. Fleig. A modelica library for simulation of electric energy

storages. *8th International Modelica Conference, Dresden, Germany*, 2011.

- M.S. Elliott, C. Estrada, and B.P. Rasmussen. Cascaded super-heat control with a multiple evaporator refrigeration system. *American Control Conference, San Francisco, USA*, 2011.
- M. Gräber, K. Kosowski, C. Richter, and W. Tegethoff. Modelling of heat pumps with an object-oriented model library for thermodynamic systems. *Mathematical and Computer Modelling of Dynamical Systems*, 16:195–209, 2010. doi:10.1080/13873954.2010.506799.
- P. Grosdidier, M. Morari, and B.R. Holt. Closed-loop properties from steady-state gain information. *Industrial and Engineering Chemistry Fundamentals*, 24:221–235, 1985. doi:10.1021/i100018a015.
- J.B. Jensen and S. Skogestad. Optimal operation of simple refrigeration cycles. part i: Degrees of freedom and optimality of sub-cooling. *Computers and Chemical Engineering*, 31: 712–721, 2007. doi:10.1016/j.compchemeng.2006.12.003.
- D. Limperich, M. Braun, and G. Schmitz. System simulation of automotive refrigeration cycles. *4th International Modelica Conference, Hamburg, Germany*, 2005.
- A.A. Pesaran, M. Keyser, K. Smith, G.H. Kim, and S. Santhanagopalan. Tools for designing thermal management of batteries in electric drive vehicles. *Large Lithium Ion Battery Technology & Application Symposia Advanced Automotive Battery Conference, Pasadena, USA*, 2013.
- C. Richter. *Proposal of New Object-Oriented Equation-Based Model Libraries for Thermodynamic Systems*. PhD thesis, Technische Universität Braunschweig, 2008.

- F. Schedel, G. Suck, S. Försterling, W. Tegethoff, and J. Köhler. Effizienzbewertung von wärmepumpen in hybridfahrzeugen mit hilfe der verlustbasierten modellierung von scrollverdichtern. *DKV-Tagung, Hannover, Germany*, 2013.
- C. Schulze. *A Contribution to Numerically Efficient Modelling of Thermodynamic Systems*. PhD thesis, Technische Universität Braunschweig, 2013.
- S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. Wiley, 2007.
- M. Titze, N. Lemke, A. Hafner, and J. Köhler. Entwicklung und simulation luftaufbereitungs- und kälteanlageanlage im supermarket mit wärmerückgewinnung. *DKV-Tagung, Hannover, Germany*, 2013.
- A. Varchmin, M. Gräber, W. Tegethoff, and J. Köhler. Superheat control with a dynamic inverse model. *10th International Modelica Conference, Lund, Sweden*, 2014.
- V.V. Viswanathan, D. Choi, D. Wang, W. Xu, S. Towne, R. Williford, J.-G. Zhang, J. Liu, and Z. Yang. Effect of entropy change of lithium intercalation in cathodes and anodes on li-ion battery thermal management. *Journal of Power Sources*, 195:3720–3729, 2009. doi:10.1016/j.jpowsour.2009.11.103.

Modeling the Effects of Energy Efficient Glazing on Cabin Thermal Energy & Vehicle Efficiency

A. S. Gravelle* Dr S. Robinson* A. Picarelli †

*Jaguar Land Rover Plc, JLR Research, Vehicle Efficiency, United Kingdom

† Claytex Services Limited, Edmund House, United Kingdom

Abstract

Automotive manufacturers are continually seeking to improve overall vehicle efficiency, one particular area of high energy consumption is the vehicle's HVAC system which can have a significant impact on fuel economy or range in electrically powered vehicles.

Presented in this paper is the work undertaken to understand the ability to model an automotive cabin for a luxury SUV in the Modelica environment including how energy efficient glazing can be modelled to determine improvements in heating or cooling efficiency at extreme ambient temperatures which will have an effect on fuel economy. The effect of air conditioning systems on fuel economy are typically not measured on cycle therefore the real world effect on energy consumption should be quantified.

The whole vehicle model and its sub-systems including the cabin and HVAC models are built using the Dymola (DYnamic MOdelling LABoratory) multi-domain physical systems engineering tool, the modelling approach to each subsystem will be discussed in this paper. The air conditioning system model has been created using 1d thermo-fluid physical models. The cabin has been modelled as a

multi-zone 1d thermo-fluid model with layering effects.

1. Introduction

1.1. Project Background

Typically in global climates where high ambient temperatures and high solar irradiance exist temperatures inside the automotive cabin can reach as high as 60°C after soaking the vehicle. Therefore the energy consumption required to bring the cabin temperature down to a comfortable level (between 20-25°C) can be significant. In order to do this the load on the air conditioning compressor is high enough to have a big impact on fuel economy.

The pulldown test is typically used to correctly size the HVAC components by applying the highest load to the AC system. Incorrect sizing of the HVAC system has a negative effect on thermal comfort and energy consumed when installed in the vehicle. If a vehicle has been soaked in a hot climate between 12pm-3pm this has a significant impact on pulldown energy requirements and time to comfort in the cabin. The final soak temperature is primarily dominated by:

surface area of the glazing, cabin volume, solar properties of the glazing, thermal conductivity & specify heat capacity of materials and the amount of solar radiation from the sun. The pulldown test consists of three 30 minute sections at: 50kph, 100kph and zero vehicle speed with idle engine speed.

In the case of low emissivity (lowE) or Infra-red reflective (IRR) glazing these have a positive effect on reflecting long wave radiation and the final soak temperature will be lower and therefore pulldown energy requirement and time to comfort will be reduced. As a result the AC system may be downsized due to the cabin now requiring significantly less energy to get to the optimum temperature.

For the models developed a baseline vehicle was developed with standard production glazing and validated against physical climatic wind tunnel test data, therefore results generated with energy efficient glazing were from the model only.

The pulldown test was run with 1000W/m² irradiance from solar lamps directly above the test vehicle, whereas in the warmup test no solar irradiance was applied to the vehicle. 43⁰C and -18⁰C temperatures were selected to be representative of the extremes of temperatures experienced across the globe. The AC compressor on the test vehicle was fitted with a specially made torque transducer so that changes in compressor torque related to cabin thermal energy could be related back to fuel consumption in the test results.

2. Efficient glazing effect on energy consumption

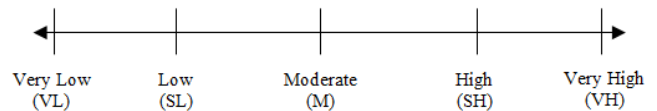
2.1. Baseline and energy efficient glazing parameters

For assessing the effect of glazing and insulation on occupant comfort and energy consumption in the modelling environment, relevant input parameters were required for the glazing model in Dymola including: transmission, absorption and emissivity characteristics. Glazing parameters are calculated directly from the supplier spectral data and conductive properties.

Table 1: Glazing input parameters for Dymola model

Baseline Car	lwef	swaf	swtf
Front Windshield	H	M	M
Panoramic Roof	H	H	L
Front Side Window	H	M	M
Rear Side Window	H	M	M
Rear Windshield	H	M	M
Prototype Car	lwef	swaf	swtf
Front Windshield (IRR)	M	L	L
Panoramic Roof (LowE)	L	H	VL
Front Side Window (LowE)	L	M	M

Scale:



- lwef* = Long wave emission factor
- k* = Thermal conductivity [W/m.K]
- swaf* = Short wave absorption factor
- swtf* = Short wave transmission factor

As can be seen from the data in Table 1.0 lowE coated glazing has a much lower emissivity value compared to standard glazing which limits the infra-red heating into the cabin. The IRR windshield has a higher emissivity than with LowE coatings but the reflective capability of long wave radiation is much better in IRR glazing,

IRR glass also has a much lower transmittance of heat into the cabin.

In both vehicle configurations the same glazing was used for the rear side windows and rear windshield as well as considering the effect that glazing targets lower solar energy transmission. Glazing that has a reduced U value was also considered, U value being the amount of heat loss per m² of material:

$$R_{th} = \frac{t}{\lambda} \quad U = \frac{1}{R_{th}}$$

R_{th} = Thermal resistance (m²/K)/W
t = Thickness of the glazing (m)
λ = Thermal conductivity (W/m.K)
U = U value (W/m².K)

U value is directly linked to thermal conductivity therefore reducing it should minimise the impact of solar gain during hot climates and prevent loss of heat from the cabin in cold climates. For simulations where low U value glazing is considered, a fixed value of 1.4W/m².K has been used.

2.2. Effects of glazing properties on the cabin environment

In testing both vehicle configurations (baseline & advanced glazing) an overall constant solar load of 1000W/m² was used

to mimic the typical values for maximum direct solar radiation (excluding the scattering effect). In reality the solar load is not constant but varies with factors such as: time of day, latitude, altitude, time of year, azimuth angle and vehicle tilt angle. The largest contributor to heat flux into the cabin is in the infrared section of the spectral curve. Therefore glazing that targets filtering of the infrared portion of the spectrum are most beneficial to lowering cabin soak temperatures. Figure 1.0 displays the comparison in solar transmittance of the baseline and LowE glazing for the front side windows used in the test vehicle.

The data shows that the transmittance of the infra-red portion of the spectrum above 800nm is significantly lower in the LowE glazing compared with the baseline glazing.

The infra-red portion of the spectrum contributes to a significant proportion of cabin heating. The coating side reflectance is significantly higher in the LowE glazing meaning less solar load is emitted into the cabin. Spectral data for the IRR windshield used in the prototype vehicle shows the percentage of transmitted light at the infra-red section of the spectrum is very low (0.6-0.01%).

Figure 1: Expected solar transmittance for LowE side windows & baseline (standard) side windows

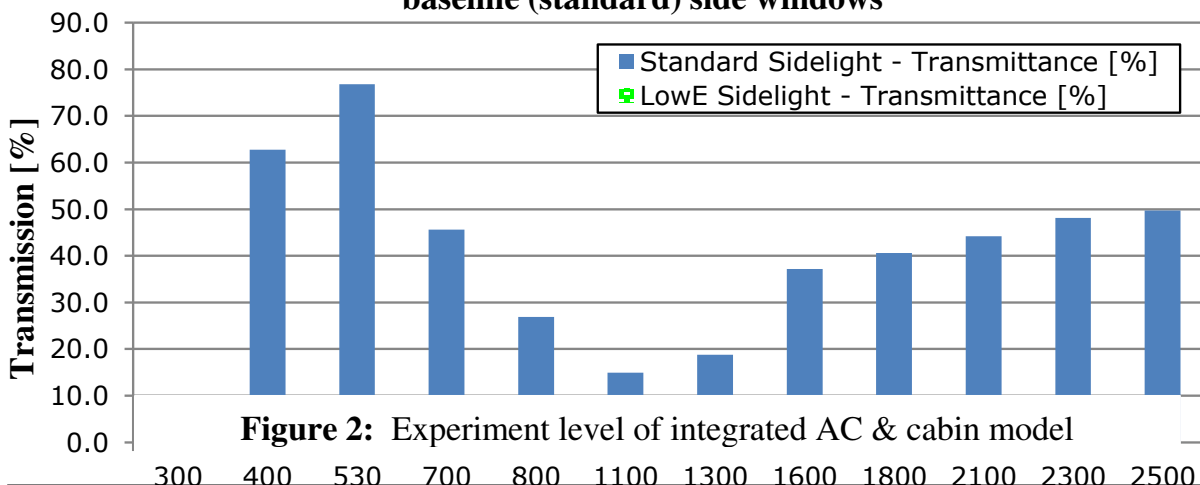


Figure 2: Experiment level of integrated AC & cabin model

The main outcome of the simulations is to compare the effectiveness of LowE and IRR glazing to test data. Typically LowE glazing is able to absorb more thermal energy and emit less radiation however IRR glass may have lower absorption capability but are able to reflect a significantly higher proportion of thermal energy which needs to be compared.

3. The HVAC & cabin simulation model

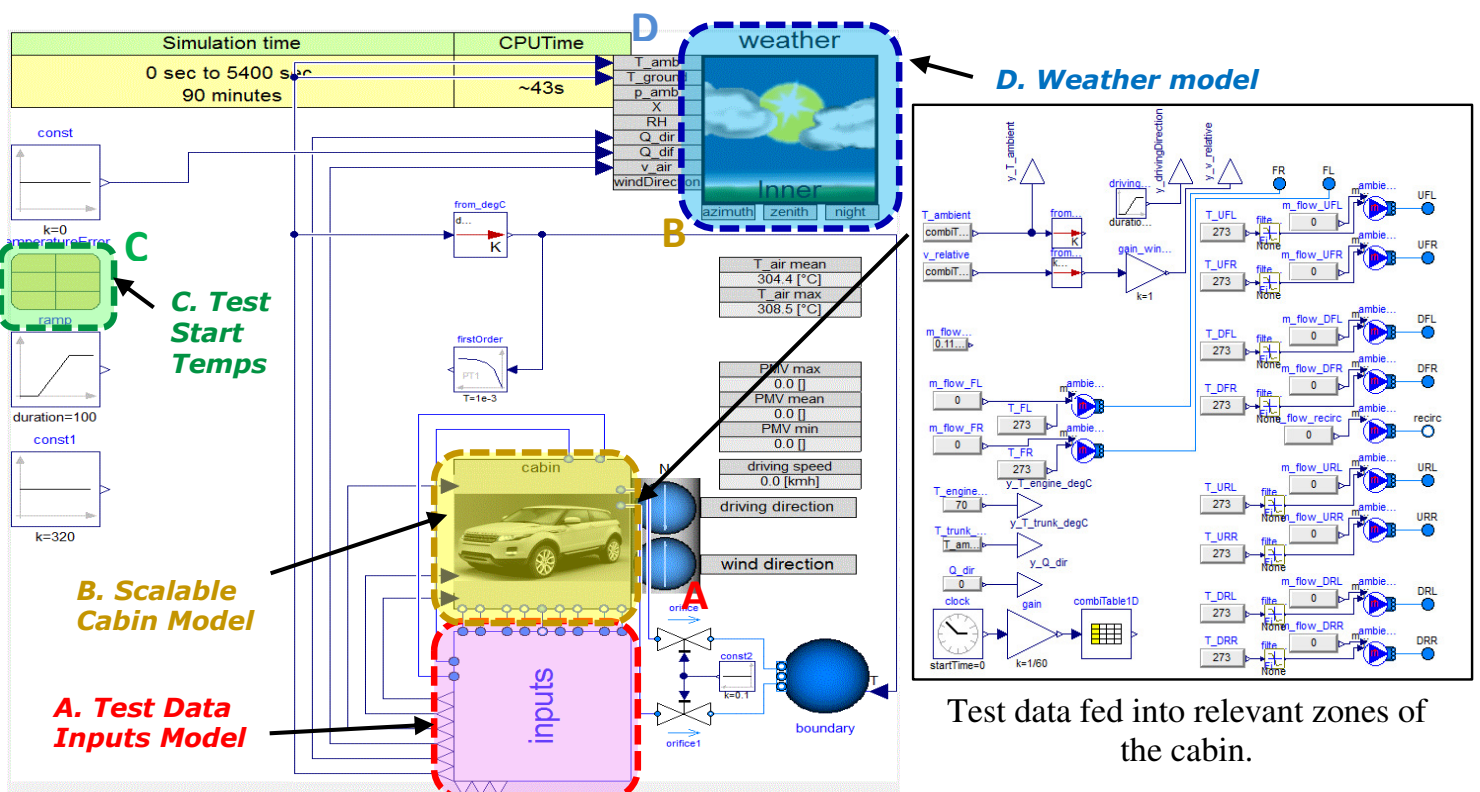
3.1. Multi-zone cabin model

The first step was to develop a physical model of the cabin and the AC system for the test vehicle being used and parameterise each model with data that is representative to the test vehicle. The cabin model was developed around an 8 zone (4 upper and 4 lower occupant zones) cabin from the Dymola Human Comfort library with some modifications to accommodate some of the parameters required for the vehicle.

Figure 2.0 shows the experiment level (top layer) of the model which includes the test data inputs which is fed into the cabin zones, a weather model transmitting ambient conditions to the outside panels of the car and initialization conditions.

There were two versions of this model developed, one that allowed the vent flows from test data to be directly used in the cabin for validation of the cabin model and the 2nd included integrating the HVAC system model to the cabin so that the vent flows are generated by the model and power consumption of the AC compressor can be determined. The AC system was used so that optimisations could be run looking at the effect of different parameters on the cabin temperature in different zones.

The inputs block which feeds the test data into the cabin model includes data such as: engine temperature, trunk temperature, solar load, zonal temperatures and ambient temperature. The experiment also shows two orifices connected to a fluid boundary seen at the bottom right of the screenshot which represent the cabin air exhausts at the back of the car.



Test data fed into relevant zones of the cabin.

The weather model feeds in the external ambient conditions to the outside surfaces of the glass and door panels such as: temperature, humidity air velocity & solar conditions.

A simplistic diagram of the cabin model itself is displayed in Figure 3.0 where in the centre of the model is the air exchange between zones. Vent air flows are fed in typically from the front occupant volumes and air exchange is permitted between upper and lower zones and travels in 1D across the volumes. Each zone has a port volume and a heat port, the volume represents the zonal cabin volume in that region and the heat port is connected to a heat sink which connects directly to the glazing & trim partitions.

The cabin volumes can be simplistic (1 upper and lower zone only) or very detailed (12 zone cabin with 1 x upper, middle & lower zone per occupant space).

The cabin model allows for calculation of thermal comfort where for each 'zone column' there is a comfort analysis in that region i.e. for head, chest or legs using parameters such as skin temperature, metabolic rate and clothing. For each of the partitions in the cabin model data records are used to capture the glazing and trim material thermal properties and also the spectral properties if relevant/available. The values detailed in Table 1.0 are applied here and also the azimuth and tilt angles of the partition as well as the positioning and thickness of each relative material within the partition stack.

Due to the complexity of modelling the characteristics of the behaviour of glazing with LowE coating or an IRR layer, for the lowE and IRR glazing the entire glass was modelled with one set of properties as a single layer for both inside and outside surfaces. The trim partitions such as doors, ceiling and floor used multiple layers depending on the structure.

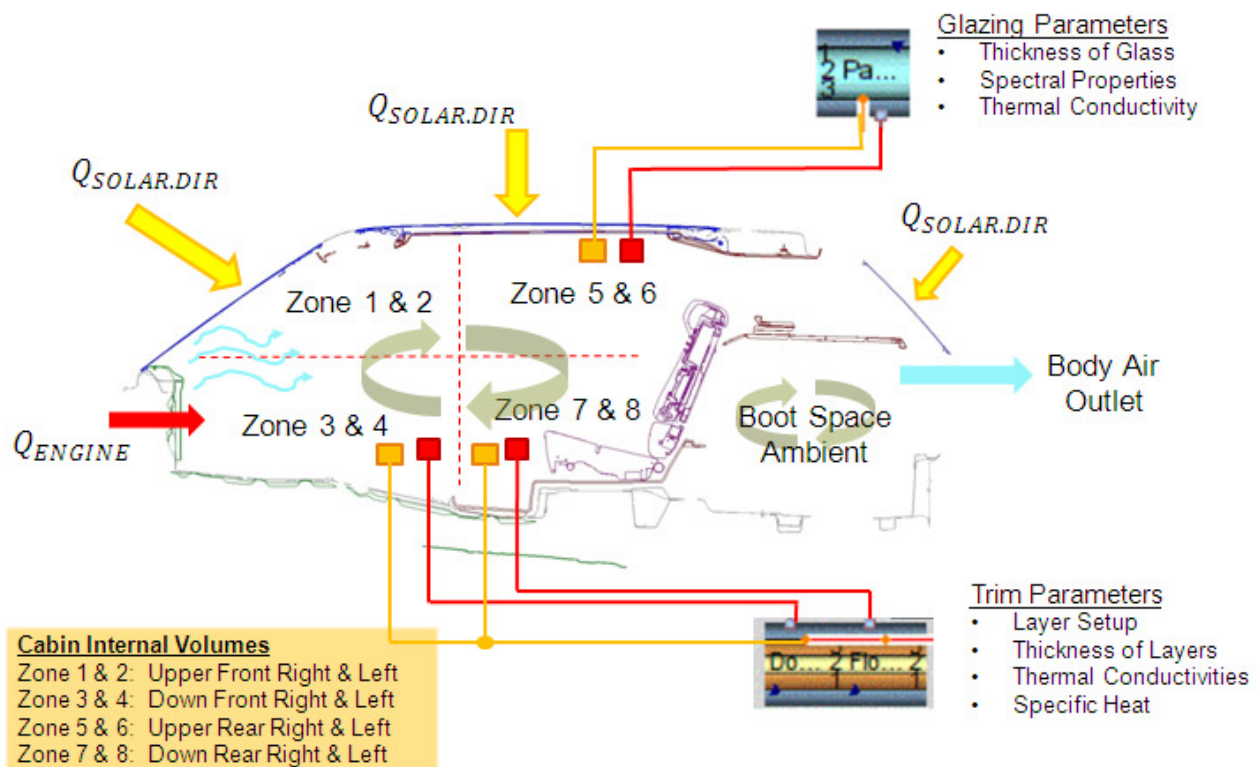


Figure 3: Schematic of cabin model with connected partition models

The thermal energy gain into the cabin is a combination of the direct solar transmittance [1], radiant heat release [2] and convective heat transfer [3], categorized by the below equations:

$$Q = (G_{dir} + G_{dif}) * swt f \quad [1]$$

$$Q = \epsilon\sigma A(T_2^4 - T_1^4) \quad [2]$$

$$Q = hA(T_2 - T_1) \quad [3]$$

3.2. Air conditioning system model

The AC system model was built up using templates from the Dymola Air Conditioning library and was fed with the ambient temperature, condenser airflow and initialization values for the AC compressor. The condenser, evaporator and thermal expansion valve ‘TXV’ are modelled physically with geometrical data for the components and the boundary conditions for the condenser and evaporator are the refrigerant states in and out of the component and the ambient conditions either side of the heat exchanger.

For iterations using the AC loop the cabin zonal temperatures are derived from the evaporator air off temperature modelled within the AC loop and valves are used to represent the relevant air duct paths into the cabin.

The mean cabin temperature was the average of all 8 zonal temperatures which applies to both test data and simulations. The relative humidity of the ambient air also has a large effect on the load of the AC system which is considered in the models. The AC compressor is of variable displacement and uses tabular efficiencies to calculate the mass flow and power consumption of the compressor:

- **Volumetric** – Volumetric efficiency of the compressor.
- **Isentropic** – Relating to enthalpy changes across the compressor.
- **Effective** – Mechanical Efficiency of the compressor

The TXV was modelled using 4 quadrant valve data and based around a TXV within the Dymola AC library. Data from the supplier was used to parameterise the TXV.

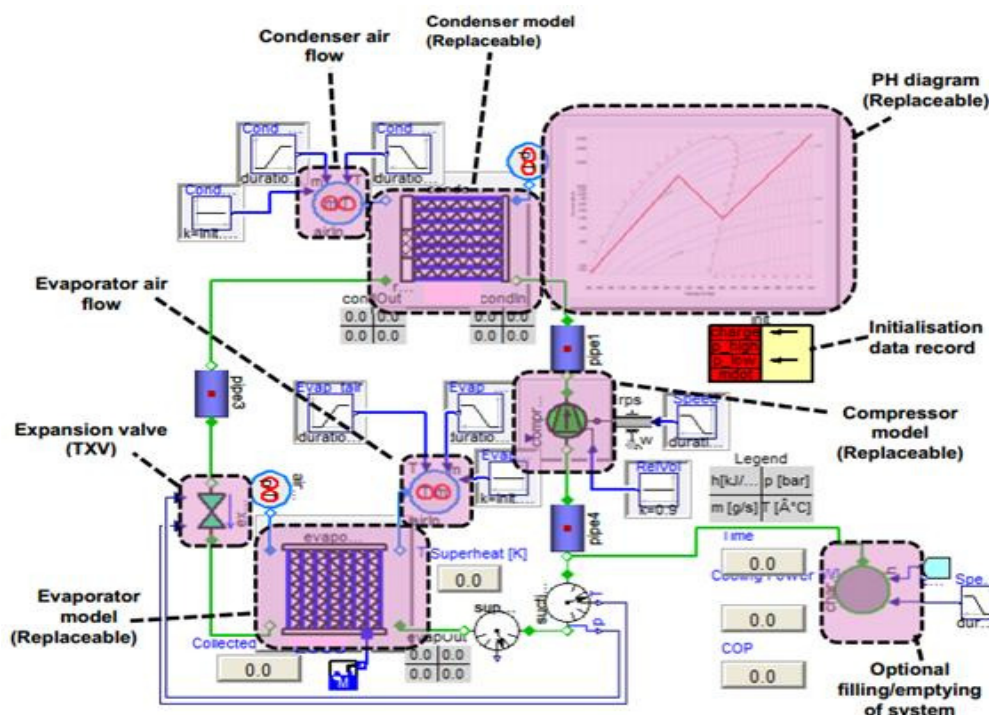


Figure 4: Air Conditioning System Model

One of the biggest limitations found when modelling the compressor was the ability to parameterize the model at part load conditions due to the difficulty in obtaining data with the required boundary conditions during testing.

The compressor is of variable swash plate angle type such that the displacement can be varied for different load conditions and to prevent icing on the evaporator as opposed to a de-clutching method. Typically the displacement of the compressor reduces when the ratio of pressures between suction and discharge reduce which is controlled by the ambient conditions.

The displacement of the compressor was varied via the use of a PID controller which uses the target evaporator air-off temperature or cabin temperature as the SP value and the model temperature as the PV value to adjust displacement.

3.3. Vehicle model

In order to assess the effect of energy consumption from the compressor torque relating to vehicle traction power a vehicle model which integrates the entire AC loop and cabin was generated. Originally a detailed engine warm-up model was included, however these models significantly increased simulation time and were excluded, Fig 5 shows the model that was used for analysis of fuel economy during the climate cycle assessment.

Driver Model: The driver model feeds in the test cycle profile speed, required accelerator pedal position and brake pedal position to the control bus. Any particular drive cycle can be used, i.e. NEDC, Artemis Urban, WLTP.

Engine Model: Is table based and uses MEP and BSFC table data to calculate the fuel flow which is dependent on throttle angle and crank speed and is controlled by the engine controller (ECU)

Transmission Model: Six speed automatic gearbox with torque converter and lockup clutch.

Chassis Model: A simplified model which includes the final drive ratio, friction brakes, wheels, vehicle mass, vehicle resistance (including aero and rolling resistance).

HVAC model: The HVAC model consists of the multi-zone cabin and AC loop where the AC compressor is connected to the accessory flange of the engine. Gearing has been used to represent the AC pulley ratio in relation the engine crankshaft.

Controllers: The engine controller uses a fuelling strategy for idle, overrun and max engine speed and the transmission controller determines the upshift and downshift points based on transmission speed as well as when to engage or disengage the lock-up clutch.

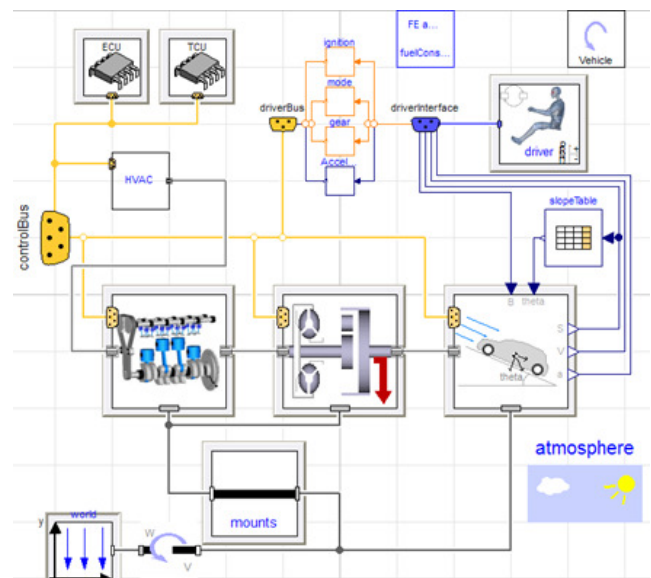


Figure 5: Vehicle Simulation Model

4. MODEL RESULTS & ANALYSIS

4.1. Correlation of Dymola model to test data

Once the entire cabin and air conditioning model was developed, a comparison of the baseline model with the test data was conducted. A comparison of the final temperatures between test data and simulation is as follows:

- Pulldown: 1.5°C
- Warmup: 2.9°C
- 43°C Soak: 0.5°C

At the start of the pulldown the cabin temperature drops very quickly due to the AC compressor running at very high displacement to be able to achieve the set point temperature. Towards the end of the pulldown the cabin temperature increases quite considerably, this is because the engine is running at idle speed with the vehicle stationary therefore the rotational speed of the compressor is lower and the airflow over the condenser is significantly reduced and thereby yields reduced cooling power. For the warmup test the heat rejection from the engine is much lower at idle so the coolant temperature passing through the heater core is much lower resulting in a temperature drop.

Figure 7 highlights simulation data for the differences in cabin temperature between different glazing types with a 43°C ambient applied, the data in this graph is purely simulation based. The reduction in cabin temperature with energy efficient glass fitted to the vehicle results in a quicker time to comfort than if standard glass were fitted to the car.

The results from the pulldown can be seen in Figure 7 and shows that applying specialised glass coatings has a big effect on lowering cabin temperature.

The most efficient setup in pulldown is to use IRR glazing with a low U value to limit solar gain and conductive heat transfer into the cabin.

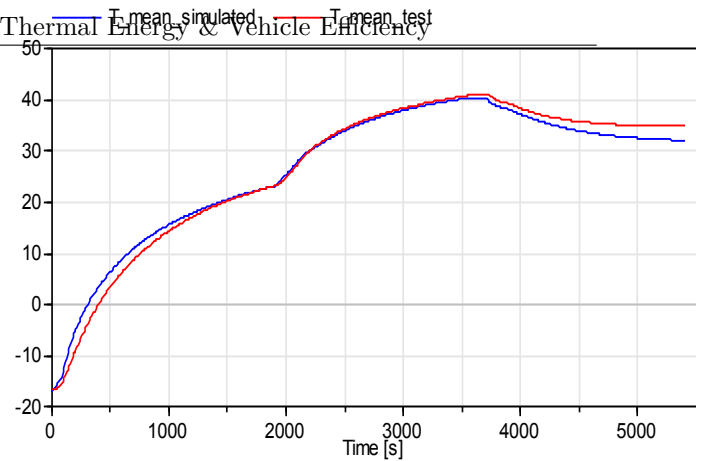
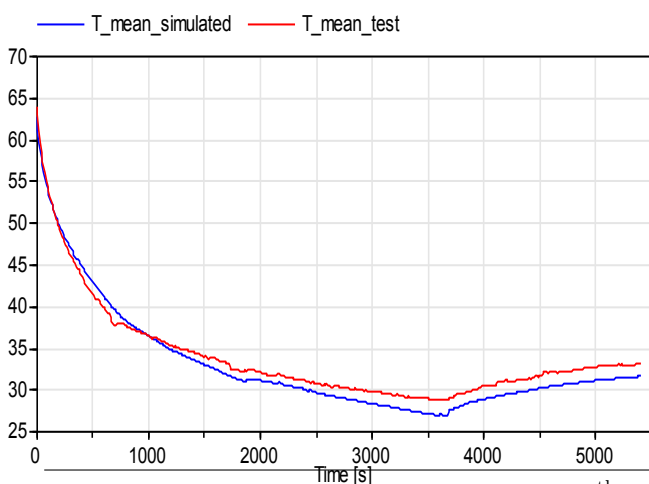


Figure 6: Comparison of test data vs. simulation data for pulldown & warmup tests

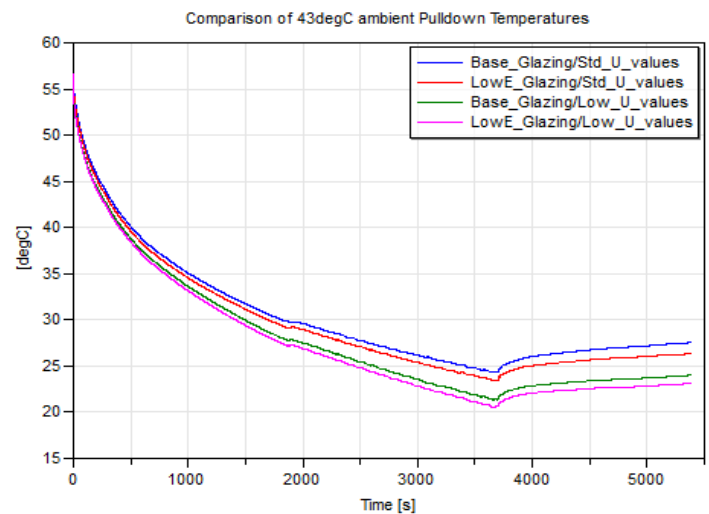


Figure 7: Comparison of various glazing types and effect on cabin temperature

4.2. Simulation results – Energy consumption

Figure 8 compares the vehicle running a pulldown with standard and energy efficient glazing for two different ambient temperatures. The energy delta of the AC compressor between standard and efficient glazing is greater at the higher ambient temperature. Therefore suggesting the greatest benefit of LowE and IRR coated glass is at the extremes of ambient temperatures.

Table 5.0 shows the associated benefits in fuel economy from a conventional diesel powered vehicle during a pulldown test in a 43°C ambient temperature with 1000W/m² of solar load.

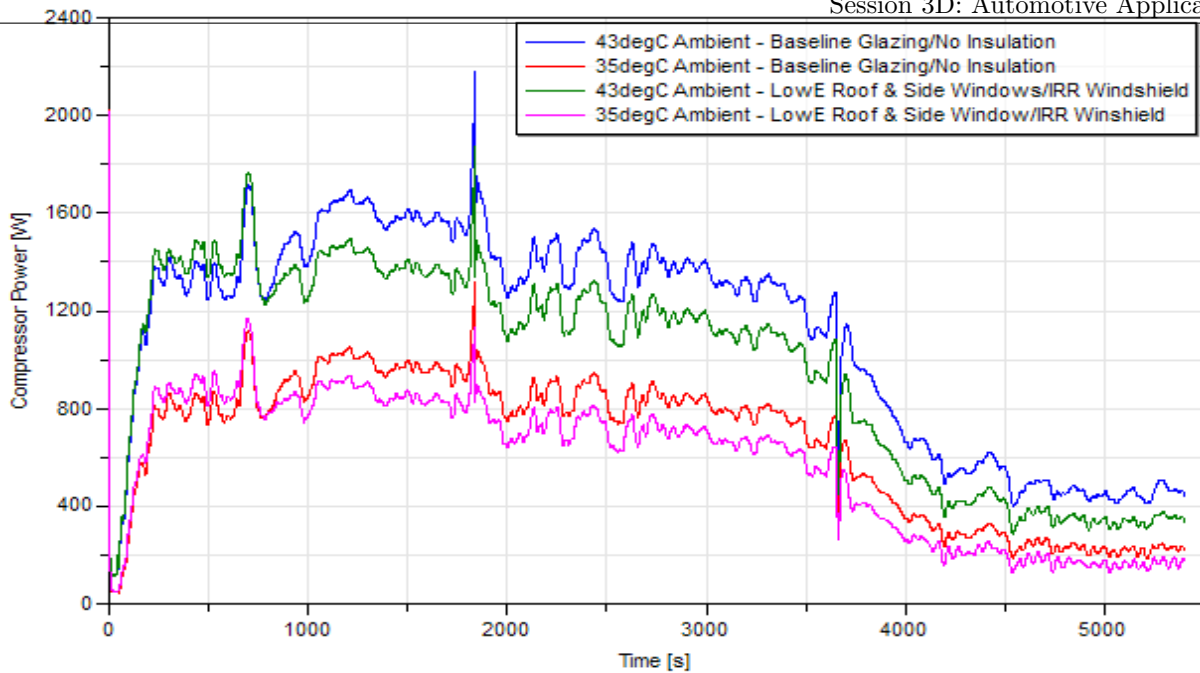


Figure 8: AC compressor torque comparing standard vs. efficient glazing

As can be seen at higher the ambient temperature the glazing has a bigger benefit in terms of fuel economy.

Condition	FE (% gain)
50°C Ambient benefit	3.3
43°C Ambient benefit	2.7
35°C Ambient benefit	1.4

Table 5: Fuel economy gains for energy efficient glazing

The benefit during the 1st 30 minutes is greater at 43°C than at 50°C due to the compressor running at full displacement for the 50°C test for both standard and energy efficient glazing, which means there is no FE benefit during the 1st 30 minutes for all configurations.

One observation from the simulations was that the highest contributor of heat flow into the cabin is from the front windshield and the panoramic roof due to their tilt angle relative to the sun, therefore optimising glazing in these areas has the biggest impact on reducing the heat flux into the cabin and reducing the load on the AC compressor.

The glazing has been considered the path of lowest thermal resistance in terms of conduction due to its thickness, therefore applying very low U values to the glazing should have a significant impact.

After generating the base cabin, HVAC and vehicle models it has been concluded that Dymola provides a model with suitable fidelity to estimate the energy benefit of novel cabin technologies. The multi-domain systems integration approach to concept studies that

Dymola provides allows the whole system to be evaluated and optimised. Understanding of the physical interaction between these systems is therefore possible. The systems integrated in this study include cabin and occupant models, AC loop, vehicle powertrain and environment. The following conclusions are established from this work:

1. A high fidelity model of the AC system and cabin model has been developed which allows replaceable components so it can be configured for use in other vehicle platforms.
2. The baseline model correlates well with the baseline test data which was used for validation purposes, therefore the model can be used as a development tool for the vehicle efficiency research team to use and potentially pass onto the mainstream.
3. Simulations show that the glazing in the panoramic roof and front windshield had

- the largest effect on solar gain and fuel economy producing FE savings as high as 5% for a 43°C pulldown depending on conditions.
4. The reduction in cabin temperature for energy efficient glazing compared to baseline windows for a 3 hour solar soak was in the order of 3-4°C, also for the pulldown this allowed a 3°C reduction in cabin temperature.
5. For the -18°C warmup test the LowE glazing alone had no impact on increasing the average cabin temperature, reducing the U value of the improved the warmup capability of the cabin. However the mode of heat transfer in this case is mostly convective.
- Requirement in a Small Car” SAE Paper 2010-01-0803, April 2010
- [5] S Gasworth, T Tankala, “Effect of Glazing Thermal Conductivity on Cabin Soak Temperature” SAE Paper 2012-01-1207, April 2012
- [6] T Han, Kuo-Huey Chen, “Assessment of Various Environmental Thermal Loads on Passenger Compartment Soak and Cool-down Analyses” SAE Paper 2009-01-1148,
- [7] D Turler, D Hopkins, H Goudey, “Reducing Vehicle Auxiliary Loads Using Advanced Thermal Insulation and Window Technologies” SAE Paper 2003-01-1076

6. ACKNOWLEDGMENTS

The authors would like to thank all the people that helped generate the content of this paper: Dr. Jonathan Parsons MInstP for providing glazing spectral properties calculated from supplier data, MIRA test facility center for experimental data and Claytex services for modeling support on the project.

7. BIBLIOGRAPHY

- [1] Michigan Scientific Corporation, “User Manual for Pulley Torque Measurement System”, Milford, MI, USA, September 2013
- [2] Green Rhino Energy, “Defining Standard Spectra for Solar Panels”,
<http://www.greenrhinoenergy.com/solar/radiation/spectra.php>
- [3] A. S. Gravelle, “A multi-domain thermo fluid approach to optimising HVAC systems”, IMA conference proceedings, Engineer’s House, Bristol, September 2014
- [4] S Shendge, P Tilekar, S Dahiya and S Kappor, “Reduction of MAC Power

A Framework for Nonlinear Model Predictive Control in JModelica.org

Magdalena Axelsson¹ Fredrik Magnusson² Toivo Henningsson¹

¹Modelon AB, Lund, Sweden, {magdalena.axelsson, toivo.henningsson}@modelon.com

²Department of Automatic Control, Lund University, Sweden fredrik.magnusson@control.lth.se

Abstract

Nonlinear Model Predictive Control (NMPC) is a control strategy based on repeatedly solving an optimal control problem. In this paper we present a new MPC framework for the JModelica.org platform, developed specifically for use in NMPC schemes. The new framework utilizes the fact that the optimal control problem to be solved does not change between solutions, thus decreasing the computation time needed to solve it. The new framework is compared to the old optimization framework in JModelica.org in regards to computation time and solution obtained through a benchmark on a combined cycle power plant. The results show that the new framework obtains the same solution as the old framework, but in less than half the time.

Keywords: Nonlinear Model Predictive Control, JModelica.org, Optimization, IPOPT

1 Introduction

Model Predictive Control (MPC) is an optimization-based control strategy based on the repeated on-line solution of an open-loop optimal control problem at discrete time points. Feedback is incorporated by measuring the state at each of these discrete timepoints and using the measured state as the initial state in the optimal control problem. From each optimization the first input in the optimal control sequence computed is applied to the system. Two of the main advantages of MPC compared to other control methods are that

- it easily extends to multivariable systems with multiple inputs and outputs.
- it intrinsically handles constraints on all system variables.

In general, one distinguishes between linear and nonlinear model predictive control (LMPC/NMPC). In the case of linear MPC, where the system model and any constraints imposed upon the system are linear and the cost is quadratic, the optimal control problem can be

cast as a quadratic program. Quadratic programs can be solved efficiently on-line. In case of nonlinear MPC, the optimal control problem is instead cast as a NonLinear Program (NLP), which is more computationally demanding to solve. The long computation time of the optimization, along with the risk that sometimes an optimal solution is not found at all, are two of the main limiting factors for successful application of NMPC in industry. (Allgöwer et al., 2004).

JModelica.org is an open-source platform for simulation, optimization and analysis of complex dynamic systems described by Modelica models (Åkesson et al., 2010). In recent research its use has been proposed for the solution of the optimal control problem for NMPC applications in several different fields, including (Cavey et al., 2014) where a JModelica.org/NMPC scheme was successfully implemented to control the heating system in a building, (Berntorp and Magnusson, 2015) where the use of JModelica.org was proposed to solve the NMPC optimal control problem in a hierarchical predictive control scheme for the lane keeping of a vehicle and (Larsson et al., 2013) where a case study of the start up of a combined cycle power plant using a JModelica.org/NMPC scheme was made. Features and performance for NMPC application using JModelica.org has been evaluated in (Hartlep and Henningsson, 2015).

The optimization algorithm in JModelica.org is currently embedded into an open-loop framework, which is well suited for solving dynamic optimization problems once. This paper describes a new optimization framework, the *MPC framework*, developed specifically for the repeated solution of the optimal control problem in NMPC schemes (Axelsson, 2015). The new MPC framework is built around the same optimization algorithm as the open-loop framework, but for efficiency it exploits the fact that the optimal control problem to solve has the same structure in each consecutive optimization. The main goals of the new MPC framework has been to decrease the average computational time for one optimization as much as possible while streamlining the setup of NMPC schemes, making JModelica.org faster and easier to use for NMPC applications.

The rest of the paper is outlined as follows. Section 2 gives general background on Nonlinear Model Predictive Control and optimization using JModelica.org. Section 3 presents the new MPC framework implemented in this paper. Section 4 compares the MPC framework to the existing open-loop framework in terms of performance on an NMPC setup of a combined cycle power plant. The section also evaluates the effects of warm starting the NLP solver. Finally, Section 5 summarizes the paper and further work discussed.

2 Background

2.1 MPC

This subsection presents the type of optimal control problems that are considered in this paper, and a basic MPC control algorithm. A common problem in MPC and how it can be solved is also discussed.

2.1.1 Optimal Control Problem

An optimal control problem includes a model of the system that is to be controlled, an objective function and, if desired, constraints on variables in the system.

The objective function, also commonly referred to as the cost function, expresses what is to be minimized in the optimization. For MPC problems the objective function is typically formulated to penalize deviations of some variables from their set points. The variables that have set points are called the *controlled variables* and may be any of the different types of variables in the system. For ease of notation we introduce w as the controlled variables

$$w = (x_{\text{controlled}}, y_{\text{controlled}}, u_{\text{controlled}}) \quad (1)$$

where $x_{\text{controlled}} \subset x, y_{\text{controlled}} \subset y$ and $u_{\text{controlled}} \subset u$.

In practice all processes are subject to constraints. These include physical constraints, such as actuators that have a limited working range and slew rate as well as constructive, safety and environmental constraints imposed on the process to make sure it is operating in a safe and desired manner. Examples of constraints included in the second category are maximum and/or minimum levels in tanks, temperatures, pressures, flow rates etc.

The purpose of an optimal control problem is to find the input that minimizes the objective function, while upholding the constraints imposed upon the system. A typical optimal control problem for MPC applications, expressed in continuous time, can have the form

minimize

$$f(w) = \int_{t_0}^{t_f} (w_{\text{ref}} - w(t))^T Q (w_{\text{ref}} - w(t)) dt \quad (2a)$$

with respect to

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

subject to

$$F(\dot{x}(t), x(t), y(t), u(t)) = 0, \quad (2b)$$

$$x(t_0) = x_0, \quad (2c)$$

$$x_L \leq x(t) \leq x_U, \quad (2d)$$

$$y_L \leq y(t) \leq y_U, \quad (2e)$$

$$u_L \leq u(t) \leq u_U, \quad (2f)$$

$$g(\dot{x}(t), x(t), y(t), u(t)) \leq 0, \quad (2g)$$

$$G(\dot{x}(t_f), x(t_f), y(t_f), u(t_f)) \leq 0, \quad (2h)$$

$$\forall t \in [t_0, t_f]$$

where (2a) is the objective function where w are the controlled variables, w_{ref} are their set points and Q is a weighting matrix. The Modelica model, expressed by a set of Differential Algebraic Equations (DAE) describing the system dynamics, is included in (2b) where $x(t)$ are the differentiated variables, $y(t)$ are the algebraic variables and $u(t)$ are the control variables. We rely on Modelica compilers to perform index reduction and thus only consider DAE systems of index at most 1, so the differentiated variables correspond to the system state. The initial conditions (2c) define the initial state $x(t_0)$ of the system. Here x_0 are the initial condition parameters. Additionally, (2d)-(2h) are the constraints imposed upon the system, where (2d)-(2f) are variable bounds with lower limit $\{x, y, u\}_L$ and upper limit $\{x, y, u\}_U$, (2g) is path constraints and (2h) is terminal constraints. The optimal control problem is considered over a prediction horizon of $H_p = t_f - t_0$ seconds.

2.1.2 Control Algorithm

The general idea with MPC is that the optimal control problem (2) is solved on-line at each sample point t_k . The solution to (2) will determine the input that is to be applied to the system until the next sample point t_{k+1} . The time between two sample points is called the sample period.

The control algorithm states that at each sample point t_k , the following steps should be carried out:

1. Obtain an estimate of the initial state $x_{\text{est}}(t_k)$.
2. Set the initial condition parameters $x_0 = x_{\text{est}}(t_k)$, the start time $t_0 = t_k$ and the final time $t_f = t_0 + H_p$.
3. Solve (2).
4. Apply u_1 to the system, where u_1 is the first value in the resulting control sequence. Hold the input constant through the entire sample period.

To be able to solve (2) we need to know the initial state of the system. Typically however, all the states are not measurable. It is therefore assumed that a state estimator is used to estimate the initial state in step 1. A Moving Horizon Estimator is an example of an optimization-based state estimator for nonlinear systems. An MHE framework is currently being developed for JModelica.org (Larsson, 2015).

2.1.3 Constraint softening

A common problem in MPC is that (2) is infeasible for the estimated initial conditions. This can happen if the process is running close to a limit and a particularly large disturbance occurs, or if the model is not good enough and the process behaves differently than predicted. Infeasibility caused by constraint violations can be prevented by softening the constraints. This means that rather than to regard constraints as hard limits which may never be crossed, we soften them by allowing them to be crossed, but only if necessary. One way of softening a constraint is by adding a new variable, a so-called *slack variable*, to the problem. This slack variable is heavily penalized in the cost function and is defined in such a way that it needs to be non-zero if the constraint is violated. With a large enough constraint penalty this gives the solver an incentive to keep the slack variable at a small value, meaning that the original constraint is upheld (Maciejowski, 2002). The MPC framework supports automatic softening of variable bounds using this method. More details on the automatic softening will be presented in Section 3.3.1.

2.1.4 Other NMPC tools

Another framework that may be used for NMPC applications based on Modelica models is the one described in (Franke et al., 2003). The framework described in that article uses multiple shooting to discretize the problem and HQP (Franke et al.), to solve the resulting NLP. One drawback with this tool, compared to the MPC framework described here, is that it implements a quasi-Newton type algorithm, meaning only first order derivatives are utilized.

ACADO toolkit is another tool suitable for NMPC applications on embedded hardware (Houska et al., 2011). However, ACADO toolkit does not have a Modelica interface and models are instead written in C++.

2.2 Optimization in JModelica.org

This subsection presents how optimization problems are solved in JModelica.org. It briefly explains the theory of the discretization and solution process, as well as how the open-loop optimization framework in JModelica.org works.

2.2.1 Discretization

The optimization algorithm in JModelica.org can solve different types of dynamic optimization problems, including the optimal control problem for MPC applications but also parameter estimation and parameter optimization problems. The optimization problems to be solved are expressed using *Optimica* (Åkesson, 2008); a Modelica extension including language constructs to e.g. formulate the objective function and constraints.

The optimization problem needs to be discretized in order for numerical solvers to solve it. To discretize the problem we let a finite number of discrete time points on the prediction horizon represent the trajectories of all variables in the optimization problem.

The optimization algorithm in JModelica.org uses direct collocation to transcribe the infinite-dimensional optimization problem into a finite-dimensional NLP (Magnusson and Åkesson, 2015). The collocation methods supported in JModelica.org are Radau and Gauss collocation. They both start with dividing the prediction horizon into n_e *collocation elements*. In each element, n_c number of *collocation points* are placed. The total number of collocation points thus becomes $n_e \cdot n_c$, and it is in these points that we consider the optimization problem. This means that each time-dependent variable in the original optimization problem, yields a set of $n_e \cdot n_c$ optimization variables in the NLP, one at each collocation point. The collocation points approximate the system trajectories by polynomials through interpolation. This in turn means that each constraint or equation in the original optimization problem, which include a time-dependent variable, is transcribed into a set of $n_e \cdot n_c$ constraints or equations in the NLP, one at each collocation point. The structure of the resulting NLP is dependent on the structure of the original optimization problem and the collocation options chosen.

2.2.2 Solving the NLP

JModelica.org uses the third party NLP solver IPOPT (Interior Point OPTimizer) to solve the resulting NLP (Wächter and Biegler, 2006). IPOPT implements a primal-dual interior point method to find a solution to the NLP, which after transcription has the general form

$$\begin{aligned} &\text{minimize} \\ &f(z) \end{aligned} \quad (3a)$$

with respect to

$$z \in \mathbb{R}^{n_z},$$

subject to

$$z_L \leq z \leq z_U \quad (3b)$$

$$g_e(z) = 0, \quad (3c)$$

$$g_i(z) \leq 0, \quad (3d)$$

where z are the optimization variables and (3b) their bounds. All constraints have been categorized depending on whether they are equality constraints g_e (3c) or inequality constraints g_i (3d). An optimal solution to the NLP requires the Karush-Kuhn-Tucker (KKT) conditions to be satisfied (Boyd and Vandenberghe, 2004). The KKT conditions can be derived from the Lagrangian function, which is defined as

$$L(z, \lambda, \nu) = f(z) + \lambda \cdot g_e(z) + \nu \cdot g_i(z), \quad (4)$$

where $\lambda \in \mathbb{R}^{n_{ge}}$ and $\nu \in \mathbb{R}^{n_{gi}}$ are the Lagrange multipliers. The Lagrange multipliers are also treated as iteration variables in the solution process. To separate them from the optimization variables z , the Lagrange multipliers are often called the *dual* variables while z are called the *primal* variables.

As an interior point method IPOPT considers the auxiliary barrier problem formulation

$$\min_z J_\mu(z) = f(z) - \mu \sum_{i=0}^{n_z} \ln(z_i) \quad (5a)$$

$$\text{s.t. } g(z) = 0 \quad (5b)$$

where μ is the barrier parameter (Wächter, 2009). This transformation from (3) to (5) is handled internally in IPOPT and for ease of notation it has here been assumed that the variables z only have lower bounds of zero. Given a value of the barrier parameter $\mu > 0$, which tends to zero during the solution procedure, the barrier objective function J will go towards infinity if any variable z approaches its bound of zero. The initial value of the barrier parameter μ_{init} determines how far away from the constraints that the intermediate solution will be pushed. For an initial guess very close to the optimal solution, a small value of μ_{init} might decrease the iterations needed to get to the optimal solution, while for a less accurate initial guess a larger value of μ_{init} typically gives faster convergence.

Given a good enough initial guess of the optimization variables the solver will converge to a local optimal solution of the NLP. An initial guess closer to the optimum will also in most cases reduce the number of iterations needed to get there. For MPC applications, it is typically a good idea to use the solution to the last optimization as the initial guess for the next.

Since the dual variables are iteration variables as well, they also need an initial guess. IPOPT has a method to compute an initial guess for the dual variables automatically. However, in the same way as for the primal variables, it might be a good idea to use the previous result of the dual variables as initial guess instead. Providing an initial guess of both primal and dual variables is called warm starting the solver and will be evaluated in section 4.3.

JModelica.org is interfaced with IPOPT through CasADi (Computer algebra system with Automatic Dif-

ferentiation)(Andersson, 2013). CasADi is an open-source, symbolic framework for automatic differentiation. It is used in JModelica.org for two main reasons; to give all optimization variables and expressions a symbolic representation using CasADi Interface (Lennernäs, 2013) and to calculate function derivatives. Scripts for JModelica.org are written in Python.

2.2.3 Optimization framework

Solving an optimization problem using the open-loop framework in JModelica.org is done in three steps:

1. Pre-processing: In the pre-processing step, the optimization problem is transcribed into an NLP by means of direct collocation as described in the previous section. All optimization variables in the resulting NLP are given a symbolic representation using CasADi and a solver object is created and initialized.
2. Solution: The solution step is handled completely by the third-party NLP solver IPOPT and includes the iterative steps that the solver takes to find a solution to the NLP.
3. Post-processing: The NLP solver returns the result for all optimization variables in one long vector. The post-processing step includes processing the result so that it is presented to the user in a convenient way, which includes creating a result object and writing the result to file.

The total computation time to solve an optimization problem is thus the time for each of these steps combined.

3 MPC framework

This section presents the new MPC framework implemented in this paper. It includes a comparison to the open-loop framework as well as a presentation of how it is used and a few of the features included in it.

3.1 Compared to open-loop framework

The MPC framework was created to make the total computation time for solving the optimal control problem shorter, while making JModelica.org easier to use for MPC applications. The reason the computation time is shorter using the MPC framework compared to directly using the open-loop framework is that the MPC framework utilizes the fact that the structure of the discretized optimal control problem is the same in each consecutive optimization. This allows performing the discretization only once, and reusing the resulting NLP for all optimizations. Solving the optimal control problem using the MPC framework is done in these steps:

0. Initialization: In the initialization step, the optimal control problem is transcribed into an NLP as in the pre-processing step of the open-loop framework.
1. Pre-processing: The initial condition parameters as well as the start and final time of the optimization horizon are updated. A new initial guess for the optimization variables is also set.
2. Solution: The solution step includes the same things as this step in the open-loop framework, with the difference that warm start of the solver can be enabled.
3. Post-processing: All u_1 values are extracted from the result and returned to the user.

Step 0 is only done once, off-line, when an MPC object is created, while steps 1-3 are executed in each optimization. Since the time-consuming discretization has been moved to initialization the pre-processing time in the MPC framework is significantly decreased. The post-processing time is also decreased due to the MPC framework not creating a result object after each optimization but rather only returning the computed u_1 values instead.

The open-loop framework hardcodes the values of Modelica parameters, including initial conditions. To enable the update of the initial conditions in Step 1 for the NLP constructed in Step 0, the initial conditions are instead introduced as symbolic NLP parameters. Defining the initial conditions as parameters x_0 thus makes it possible to update their values between optimizations.

3.2 Example

The MPC framework includes features that simplify the use of JModelica.org for MPC purposes. After the setup, the MPC object requires very little interaction from the user as most things are handled internally. The only information that has to be supplied to the MPC object is the next initial state. The Python code excerpt below gives an example on how the MPC framework is used. Here it is assumed that the optimization problem `opt_problem`, the optimization options `options`, the sample period `sample_period` and the prediction horizon `horizon` have already been defined. A detailed description of how to do this is found in (Axelsson, 2015). The Optimica code for the benchmark system used in this article will be presented in Section 4.2.

The first line of this example shows how to utilize the support for automatically softening variable bounds, in this case for the variable `plant.sigma`. In this example, artificial measurement data is created by simulating an FMU of the system from the initial state and one sample period forward in time, with the optimal input obtained from the optimization.

```
# Define variable bounds to be softened
cvc = {'plant.sigma': 1e5}
```

```
# Create the MPC object
MPC_object = MPC(opt_problem, options,
                 sample_period, horizon, constr_viol_costs=cvc
                 )

# Set initial state
x_k = {}
for name in op.get_state_names():
    x_k["_start_"+name] = opt_problem.get("_start_"+name)

for k in range(nbr_opt):
    # Update the state and optimize nbr_opt times
    MPC_object.update_state(x_k)
    u_k = MPC_object.sample()

    # Simulate for one sample period with the
    # optimal input u_k
    sim_model.reset()
    sim_model.set(x_k.keys(), x_k.values())
    sim_res = sim_model.simulate(
        start_time = k*sample_period,
        final_time = (k+1)*sample_period,
        input=u_k, options=sim_opts)

    # Extract state values at end of sim_res
    x_k = MPC_object.extract_states(sim_res)
    # Add measurement noise to states

# Get result and extract variable profiles
opt_res = MPC_object.get_complete_results()
opt_plant_sigma = opt_res['plant.sigma']
```

3.3 Features

3.3.1 Softening Variable Bounds

The need for constraints to be softened was discussed in Section 2.1.3. The MPC framework has a method that automatically softens variable bounds. The softening is done before the discretization, and will thus be described in continuous time. For each variable bound that is to be softened, a slack variable is added to the problem formulation. That means that if a variable z has both an upper limit z_U and a lower limit z_L , the same slack variable, z_{slack} , will be used when softening both bounds. The softening is done in four steps:

1. A new input, the slack variable z_{slack} , is added to the optimization problem. The slack variable is bounded to be larger than 0 and the nominal value is set to 0.0001 times the nominal value of the base variable. That is,

$$z_{\text{slack}} \geq 0 \quad (6)$$

$$z_{\text{slack, nominal}} = 0.0001 \cdot z_{\text{nominal}} \quad (7)$$

2. The slack variable times a constraint violation penalty P_z is added to the cost function. That is, the cost function $f(w)$ is changed to:

$$f(w) + P_z \cdot \int_{t_0}^{t_f} z_{\text{slack}}(t) dt \quad (8)$$

Once discretized, this formulation will be equivalent with adding the 1-norm of the slack variable times the constraint violation penalty.

3. The old variable bounds are transformed into path constraints on the form

$$z \leq z_U + z_{\text{slack}} \quad (9)$$

$$z \geq z_L - z_{\text{slack}} \quad (10)$$

These four steps are done for all variables that have bounds to be softened. If a variable has only either an upper or a lower bound, step 3 is modified accordingly. Ideally, if the initial condition has not violated the constraint, the slack variable should be zero or very close to zero at all times. However, choosing the nominal value of the slack variable has to be done with care to avoid numerical issues. This is why we have made the nominal value of the slack variable proportional to the nominal value of the base variable. The factor of 0.0001 included in the calculation of the slack nominal value was decided through testing.

3.3.2 Unsuccessful Optimization

Since finding a solution to the NLP is not guaranteed, it is important to have a fallback method in case of unsuccessful optimization. Using the MPC framework, if the solver terminates without finding a solution to the given problem the input returned will be the second input u_2 in the input sequence of the *previous* optimization (which was successful). If the next optimization after that is unsuccessful as well, the third input u_3 in the input sequence in the last successful optimization is returned, and so on. This way of returning optimal inputs from the last successful optimization continues until the solver finds a feasible solution again, or until there are no more values in the last successful optimization to return.

This is the default fallback method in case of unsuccessful optimizations the MPC class uses. However, it is straightforward to detect if an optimization was successful or not, so it is possible for the user to create a custom fallback method instead.

3.3.3 Next initial guess

Having a good initial guess for the optimization variables is important to decrease the risk of not finding a solution and to speed up the solution process. Defining a new initial guess of the optimization variables prior to each optimization is handled internally in the MPC framework. There are three different methods of computing the initial guess in the MPC framework:

1. *Extracting it from a result object.* This method uses the same methods that are used by the open-loop framework to extract an initial guess of the optimization variables from the trajectories of a result

object. This method is quite time-consuming and requires that a result object, from which to extract the initial guess, is available.

2. *Shifting the result vector.* The NLP solver returns the solution of an optimization in one vector containing the value of each variable at each of the collocation points. The result vector is on the same form as the vector corresponding to the initial guess, but offset by one sample period in time. Looking at the result vector, this method discards all the values included in the first sample period and shifts the rest of the values to cover the voids. This means that all the values corresponding to the second sample period in the result vector will be shifted to the values corresponding to the first sample period in the initial guess vector. The values of the last sample period in the initial guess vector are all set to the value of the last collocation point from the result vector. On a uniform mesh, this method yields the same initial guess as method 1 and is less time-consuming.
3. *Using the result vector without shifting it.* This method sets the new initial guess to the result from the previous optimization directly, without shifting it. Using this method will yield the least accurate initial guess, since all values will be offset by one sample period in time, but it is the least time consuming method of the three.

The default method of computing the next initial guess in the MPC framework is method 2, mainly because it is faster than method 1 and yields a better initial guess than method 3.

3.4 Limitations

Because the MPC framework reuses the NLP for each optimization it is not as flexible as the open-loop framework. There are currently some collocation options that are not compatible or will not work as desired with the MPC framework and there are also some restrictions regarding the formulation of the optimal control problem. These are described in more detail in (Axelsson, 2015).

4 Results

4.1 Test setup

In this section we will evaluate the performance of the MPC framework through two different tests. The first test is to evaluate how the performance of the NLP solver is affected by the warm start options chosen. The second test is a benchmark where the aim is to compare the results of the MPC framework to the open-loop framework. For both tests we provide some or all of the following statistics:

- **Opt_{fail}**. The sample number of the optimizations which were unsuccessful. For IPOPT it is assumed that the return statuses 'Solve_Succeeded' and 'Solved_To_Acceptable_Level' denote a successful optimization. All other return statuses are regarded as unsuccessful.
- **Iterations**. The average number of iterations in IPOPT for one sample.
- **T_{pre}**. The average pre-processing time for one sample.
- **T_{sol}**. The average solution time in IPOPT for one sample.
- **T_{post}**. The average post-processing time for one sample.
- **T_{tot}**. The average total computation time for one sample.

All tests are run using the MA27 solver for IPOPT (HSL, 2013).

4.2 Test problem

The system we are going to evaluate the performance of the MPC framework on is a Combined-Cycle Power Plant (CCPP) (Casella et al., 2011), during start-up. The aim of the MPC controller is to take the system from an off state to full capacity. The plant is considered to be at full capacity once the evaporator pressure p has reached 8.35 MPa and the plant load $load$ has reached 100%. During the start-up there is an upper bound on the thermal stress σ in the steam turbine, which may not exceed 260 MPa. The MPC framework will soften this bound automatically as discussed in section 3.3.1. We are going to extend the model with an integrator at the input by connecting the plant load with a new state variable u and thus having \dot{u} as the input in the optimization problem. This yields a plant load which is piecewise linear, rather than piecewise constant. This also allows for setting variable bounds on \dot{u} . Variable bounds on u and \dot{u} are

$$\begin{aligned} 0 &\leq u \leq 1, \\ 0 &\leq \dot{u} \leq 0.1/60. \end{aligned}$$

The Optimica code for this system is presented below.

```
optimization Startup(objectiveIntegrand=((
  plant.p-8.35e6)/1e6)^2 + 0.5*(u-1)^2,
  startTime=0,finalTime=4000)

parameter Real sigma_max = 2.6e8;
CombinedCycle.Optimization.Plants.CC0D_WarmStartUp
  plant(sigma(max=sigma_max));
Modelica.Blocks.Interfaces.RealInput du(min=0,
  max=0.1/60);
RealConnector u(start=0.15, fixed=true, min=0, max
  =1);
```

```
equation
der(u) = du;
connect(u, plant.load);

end Startup;
```

On the first line, the keyword `objectiveIntegrand` is used to define the Lagrange part of the cost function, while `startTime` and `finalTime` denote the beginning and end of the prediction horizon. A model of the plant is instantiated, `plant`, and an upper variable bound is added to `sigma` using the keyword `max`. The following two lines show how to add variable bounds to the inputs, `u` and `du`, and how to connect them to the model. Additional constraints are not present in this example, but could be added under a new section started with the keyword `constraint`.

To emulate noise a normally distributed disturbance, with the mean 0 and the standard deviation 0.001 times the current state value, will be added at each sample point to all the states except for the extra state u .

With the addition of u as a state, and the extra input σ_{slack} which the MPC framework will add to the problem when softening the bound on σ , the resulting optimization problem has 10 states, 123 algebraic variables and 2 inputs. With the MPC and collocation options chosen, presented in Table 1, the resulting NLP has 4564 optimization variables after the discretization.

Table 1. The MPC and collocation options used for all tests on the CCPP system.

MPC options	value
Sample period	100 [s]
Prediction horizon	1000 [s]
Collocation options	value
n_e	10
n_c	3

4.3 Warm start test

The warm start test aims to evaluate whether we can improve the robustness and speed of the solver by providing an initial guess of the dual variables to the solver. The options we consider in IPOPT are 'warm_start_init_point', which indicates whether an initial guess of the dual variables will be provided by the user or should be estimated by IPOPT, and 'mu_init', which is the initial value of the barrier parameter. For the cases where an initial guess of the dual variables will be provided, the guess will be the result from the previous optimization. Note that since there is no implemented support for shifting the dual variables yet, they will be

implicitly offset by one sample period in time. The result of this test is presented in Table 2.

Table 2. Summary of the results for the warm start test. The two leftmost columns define which options were used, while the other three are the results obtained. Warm start on means that an initial guess of the dual variables was provided to the solver while warm start off means that the solver estimated it's own initial guess for them.

Warm start	μ_{init}	Opt_{fail} [k]	Iterations [nbr]	T_{sol} [s]
Off	1e-1	-	32	0.282
Off	1e-2	-	34	0.289
Off	1e-3	5, 24	32	0.282
Off	1e-4	-	34	0.282
On	1e-1	-	31	0.266
On	1e-2	-	31	0.261
On	1e-3	-	32	0.262
On	1e-4	-	30	0.259

From the data in Table 2 it can be seen that providing an initial guess of the dual variables decreases both the number of iterations needed to find a solution and the solution time slightly. The robustness also seems to be improved since optimal solutions were found for all samples in the case where warm start was on, while two unsuccessful optimizations were noted when warm start was off. The best average solution time was in the case where warm start was on and $\mu_{init} = 10^{-4}$.

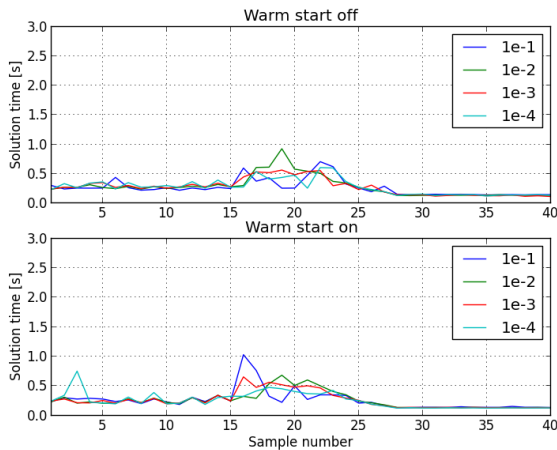


Figure 1. The solution time for each of the samples in the warm start test. The upper plot is for warm start being turned off and the μ_{init} values as specified by the legend and the lower plot is for warm start being turned on.

In Figure 1 the solution time for each of the samples is plotted for all the options tested. Since 8 different option combinations were tested, the results have been split into two separate plots, one where warm start is turned off and one where warm start is turned on. The barrier parameters impact on the solution time is especially noticeable

in the region between sample number 15 and 25, where the largest deviations are present. The overall conclusion from this test is that turning the warm start on i.e. providing an initial guess of the dual variables to the solver, has a positive effect on the solution time and robustness. This even though the dual variables provided are offset by one sample period in time. The gain of warm starting the solver might improve even more if a shift method for the dual variable was to be implemented.

4.4 Benchmark

In this section the results of using the MPC framework will be compared to the results of using the open-loop framework for an MPC setup. We will look specifically at the result trajectories as well as the different average times for one sample (pre-processing, solution, post-processing and total).

To get equivalent problem formulations the variable bound on σ is softened manually for the case where the open-loop framework is used. The manual softening is done in exactly the same way as the MPC framework does it. The collocation and MPC options chosen are the same in both cases and the resulting NLP:s shall thus be identical in both cases. For the case running with the MPC framework warm start of the solver is activated and the barrier parameter is set to $\mu_{init} = 10^{-4}$, since those were the options that gave the best results in the warm start test. The results are summarized in Table 3.

Table 3. Results from the benchmark of the CCP system.

	Opt_{fail} [k]	$T_{pre,}$ [s]	$T_{sol,}$ [s]	$T_{post,}$ [s]	$T_{tot,}$ [s]
MPC fw.	-	0.053	0.267	0.012	0.332
OL fw.	12	0.901	0.295	0.044	1.241

From the data in Table 3 it can be concluded that using the MPC framework compared to using the open-loop framework has decreased the total average computation time by 70%. This is also clearly illustrated in the total computational time per sample plot in Figure 2. Looking closer at the average times we can conclude that the majority of the time saved is in the pre-processing step, which was what we had expected since the time consuming discretization has been moved outside the MPC loop. The post-processing time is also decreased due to the MPC framework not creating a result object after each optimization.

Figure 3 shows the CCP system simulated with the optimal inputs obtained, where we can conclude that the results obtained in both cases are almost identical.

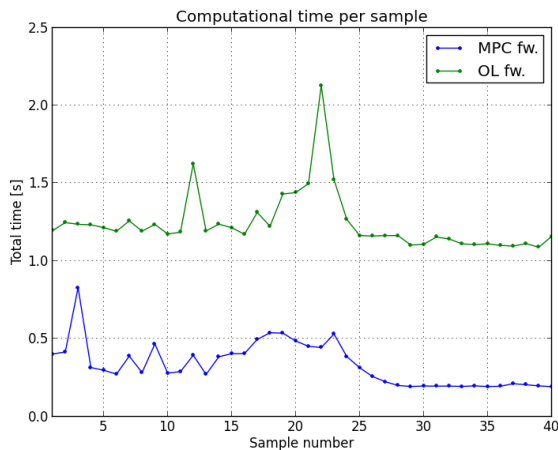


Figure 2. Total computation time for each sample in the benchmark, using the MPC framework and the open-loop framework respectively.

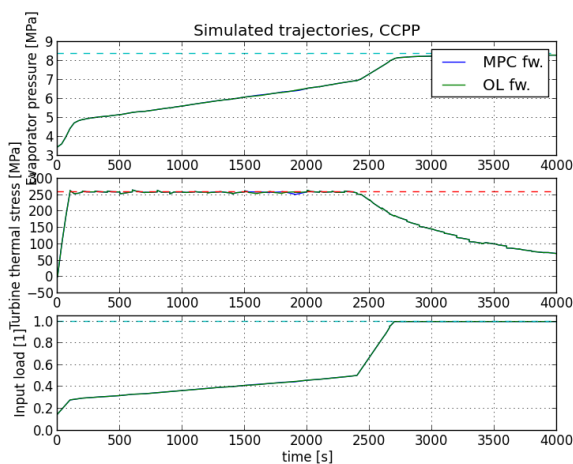


Figure 3. The CAPP system simulated with the optimal inputs obtained in both cases. The dashed cyan lines are the set points while the dashed red line is the variable bound.

5 Conclusions

This paper describes the implementation of a new MPC framework in JModelica.org, which significantly decreases the total average computation time of solving an NMPC optimal control problem. The main reason the computation time has been shortened is due to the MPC framework reusing the NLP for all optimizations, rather than creating a new NLP each time the optimal control problem is solved. For the benchmark presented in this paper, using the MPC framework compared to using the open-loop framework, the total average computation time went from 1.24 s to 0.33 s, a relative decrease of 70%. The benchmark also shows that the same results were obtained with both frameworks.

In addition to being faster than the open-loop framework, the MPC framework is also easier to use since a lot of things are handled internally. This includes the initial

guess being set and the prediction horizon being shifted automatically as well as the built in fall back method in case of unsuccessful optimization and a method that automatically softens variable bounds.

Further work includes adding support for nominal trajectories and external data, two of the collocation options that do not work correctly when reusing the NLP. Nominal trajectories are used for scaling the optimization variables and external data could be used to define set point trajectories, rather than constant set points, for the controlled variables. A shift method for the dual variables could also be implemented to, hopefully, decrease the solution time in the solver further. The automatic softening of variable bounds method could be extended to support automatic softening of constraints as well as different softening schemes.

Acknowledgments

Fredrik Magnusson acknowledges support from the Swedish Research Council through the LCCC Linneaus Center and is also a member of the eLLIIT Excellence Center at Lund University.

References

- "HSL. A collection of Fortran codes for large scale scientific computation.", 2013. URL <http://www.hsl.rl.ac.uk/>.
- Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *6th International Modelica Conference 2008*, 2008.
- Johan Åkesson, Karl-Erik Årzén, Magnus Gäfvert, Tove Bergdahl, and Hubertus Tummescheit. Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problems. *Computers and Chemical Engineering*, 34(11):1737–1749, November 2010.
- Frank Allgöwer, Rolf Findeisen, and Zoltan K Nagy. Nonlinear model predictive control: From theory to application. *J. Chin. Inst. Chem. Engrs*, 35(3):299–315, 2004.
- Joel Andersson. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, October 2013.
- Magdalena Axelsson. Nonlinear Model Predictive Control in JModelica.org. Master's thesis, Department of Automatic Control, Lund University, Sweden, August 2015.
- Karl Berntorp and Fredrik Magnusson. Hierarchical predictive control for ground-vehicle maneuvering. In *2015 American Control Conference*, 2015.
- Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

Francesco Casella, Filippo Donida, and Johan Åkesson. Object-oriented modeling and optimal control: A case study in power plant start-up. In *18th IFAC World Congress*, 2011.

Mats Vande Cavey, Roel De Coninck, and Lieve Helsen. Setting up a framework for model predictive control with moving horizon state estimation using jmodelica. In *10th International Modelica Conference 2014*, 2014.

R. Franke, E. Arnold, and H. Linke. HQP: a solver for nonlinearly constrained large-scale optimization. URL <http://hqp.sourceforge.net/>.

Rüdiger Franke, Manfred Rode, and Klaus Krüger. On-line optimization of drum boiler startup. 2003.

Christian Hartlep and Toivo Henningsson. NMPC Application using JModelica.org: Features and Performance. In *11th International Modelica Conference 2015*, 2015.

B. Houska, H.J. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

Per-Ola Larsson, Francesco Casella, Fredrik Magnusson, Joel Andersson, Moritz Diehl, and Johan Åkesson. A framework for nonlinear model-predictive control using object-oriented modeling with a case study in power plant start-up. In *Computer Aided Control System Design (CACSD), 2013 IEEE Conference on*, 2013.

Tor Larsson. Moving Horizon Estimation in JModelica.org. Master's Thesis ISRN LUTFD2/TFRT--5982--SE, Department of Automatic Control, Lund University, Sweden, 2015.

Björn Lennernäs. A CasADi based toolchain for JModelica.org. Master's thesis, Department of Automatic Control, Lund University, Sweden, June 2013.

J.M. Maciejowski. *Predictive Control with Constraints*. Prentice-Hall, 2002.

Fredrik Magnusson and Johan Åkesson. Dynamic optimization in jmodelica.org. *Processes*, 3(2):471–496, 2015.

Andreas Wächter. Short tutorial: getting started with ipopt in 90 minutes. *Combinatorial Scientific Computing (U. Naumann, O. Schenk, HD Simon, eds.)*, 34(56):118, 2009.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

A Toolchain for Solving Dynamic Optimization Problems Using Symbolic and Parallel Computing

Evgeny Lazutkin Siegbert Hopfgarten Abebe Geletu Pu Li

Group Simulation and Optimal Processes, Institute for Automation and Systems Engineering, Technische Universität Ilmenau, P.O. Box 10 05 65, 98684 Ilmenau, Germany.

{evgeny.lazutkin, siegbert.hopfgarten, abebe.geletu, pu.li}@tu-ilmenau.de

Abstract

Significant progresses in developing approaches to dynamic optimization have been made. However, its practical implementation poses a difficult task and its real-time application such as in nonlinear model predictive control (NMPC) remains challenging. A toolchain is developed in this work to relieve the implementation burden and, meanwhile, to speed up the computations for solving the dynamic optimization problem. To achieve these targets, symbolic computing is utilized for calculating the first and second order sensitivities on the one hand and parallel computing is used for separately accomplishing the computations for the individual time intervals on the other hand. Two optimal control problems are solved to demonstrate the efficiency of the developed toolchain which solves one of the problems with approximately 25,000 variables within a reasonable CPU time.

Keywords: nonlinear optimization, combined multiple shooting and collocation, symbolic manipulation, parallel computing, satellite problem, combined cycle power plant

1 Introduction

Over the last decades nonlinear model predictive control (NMPC) has been increasingly popular for the control of complex systems (Mayne, 2014). To carry out NMPC, the first step is to formulate a nonlinear optimal control problem. By using a discretization scheme over a prediction horizon, it is then transformed into a constrained nonlinear programming (NLP) problem. Finally, the realization of NMPC is made by repeatedly solving this problem online with an NLP solver which requires appropriate function values and gradients. Although many theoretical progresses on NMPC have been achieved, its implementation for real-life applications is certainly not trivial. Therefore, a toolchain is developed in this work based on open-source software tools to relieve the burdens in the implementation of NMPC.

A schematic description of implementing NMPC is

shown in Fig. 1. Based on the current process state $x(k)$ obtained through the state observer or measurement, resp., the optimal control problem is solved in the optimizer in each sample time. The resulting optimal control strategy in the first interval $u(k)$ of the moving horizon is then realized through the local control system. Therefore, an essential limitation of applying NMPC is due to its long computation time taken to solve the NLP problem for each sample time, especially for the control of fast systems (Wang and Boyd, 2010). In general, the computation time should be much less than the sample time of the NMPC scheme (Schäfer et al., 2007). Although powerful methods are available, e.g. multiple-shooting (Houska et al., 2011; Kirches et al., 2012) and collocation on finite elements (Biegler et al., 2002; Zavala et al., 2008; Word et al., 2014) with simultaneous characteristics, control parametrization (Balsa-Canto et al., 2000; Barz et al., 2012) with sequential characteristics, and quasi-sequential technique, (Hong et al., 2006; Bartl et al., 2011), the computation speed is not swift enough for very fast systems such as mechanical, electrical and mechatronic systems. Therefore, it is highly desired to further enhance the computation efficiency for solving nonlinear dynamic optimization problems.

The combined multiple-shooting with collocation (CMSC) method (Tamimi and Li, 2010) and the modified multiple-shooting and collocation (MCMSC) method (Lazutkin et al., 2014) are proved to be highly efficient. The efficiency of this method is considerably improved in this work with the following targets:

- to reduce the computation time by using symbolic methods for calculating gradients, Jacobians, and Hessians,
- to further accelerate the computation by using parallel computing facilities, especially for real-time applications.

To achieve these aims, this work develops a toolchain as described in subsequent sections. In section 2 the problem will be formulated. Section 3 illustrates the interior-point solution method with symbolic computations of first- and second-order derivatives. The

toolchain, its components, functionality, and some code examples are presented in section 4. The efficiency of the toolchain is demonstrated in section 5 by applying it to a satellite control and a large-scale dynamic optimization of a combined cycle power plant. Conclusions of the paper are given in section 6.

2 Problem description

The nonlinear optimal control problem (NOCP) reads

$$\begin{aligned} \min_{\mathbf{u}(t)} & \left\{ J = M(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t) dt \right\} \\ \text{s. t. } & \mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), t) = \mathbf{0}, \quad t_0 \leq t \leq t_f, \\ & \mathbf{x}(t_0) = \mathbf{x}_0, \quad \mathbf{x}(t_f) \text{ fixed or free,} \\ & \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t) \leq \mathbf{0}, \\ & \mathbf{x}_{min} \leq \mathbf{x}(t) \leq \mathbf{x}_{max}, \\ & \mathbf{u}_{min} \leq \mathbf{u}(t) \leq \mathbf{u}_{max}, \end{aligned} \quad (1)$$

with $t \in [t_0, t_f]$ - time, t_0, t_f - initial, final time, $\mathbf{x}(t) \in \mathbb{R}^{n_x}$ - state variable vector, $\mathbf{u}(t) \in \mathbb{R}^{n_u}$ - control variable vector, \mathbf{x}_0 - initial state vector, \mathbf{x}_f - final state vector, $\mathbf{f} \in \mathbb{R}^{n_x+n_u} \rightarrow \mathbb{R}^{n_x}$ - implicit differential equation, J - performance index with Mayer and Lagrange term $M: \mathbb{R}^{n_x+1} \rightarrow \mathbb{R}$ and $L: \mathbb{R}^{n_x+n_u} \rightarrow \mathbb{R}$, resp., belonging to corresponding function spaces, \mathbf{g} - additional equality and/or inequality constraints, $\mathbf{x}_{min}, \mathbf{x}_{max}, \mathbf{u}_{min}, \mathbf{u}_{max}$, - componentwise lower and upper bounds for states $\mathbf{x}(t)$ and controls $\mathbf{u}(t)$, resp.

Envisaging the application of the modified combined multiple shooting and collocation (MCMSC) method, a transformation of the infinite-dimensional NOCP (1) to a finite-dimensional nonlinear programming (NLP) problem is needed. Due to the multiple-shooting technique a division of the whole time horizon $[t_0, t_f]$ into N time intervals, so called shooting intervals has to be performed. The controls are assumed to be constant in each shooting interval and are parametrized, i.e. the control vector is composed as $\mathbf{V} = [\mathbf{v}_0 \ \mathbf{v}_1 \ \dots \ \mathbf{v}_{N-1}]^T$, $n_u = n_v$. The states are also discretized and parametrized at the shooting interval boundaries, i.e. the vector $\mathbf{X}^P = [\mathbf{x}_{p,0} \ \mathbf{x}_{p,1} \ \dots \ \mathbf{x}_{p,N}]$ is constructed and equality constraints for continuity reasons are taken into account. All other restrictions are correspondingly discretized.

This leads to the NLP notation

$$\begin{aligned} \min_{\mathbf{X}^P, \mathbf{V}} & \left\{ M(\mathbf{x}_{p,N}) + \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} L(\mathbf{x}(t), \mathbf{v}_i) dt \right\} \\ \text{s. t. } & \mathbf{x}_{p,i+1} = \mathbf{x}(t_{i+1}; \mathbf{x}_{p,i}, \mathbf{v}_i), \quad i = 0, \dots, N-1, \\ & \mathbf{x}_{p,0} = \mathbf{x}_0, \\ & \bar{\mathbf{g}}(\mathbf{X}^P, \mathbf{V}) \leq \mathbf{0}, \\ & \bar{\mathbf{x}}_{min} \leq \mathbf{X}^P \leq \bar{\mathbf{x}}_{max} \\ & \bar{\mathbf{u}}_{min} \leq \mathbf{V} \leq \bar{\mathbf{u}}_{max}. \end{aligned} \quad (2)$$

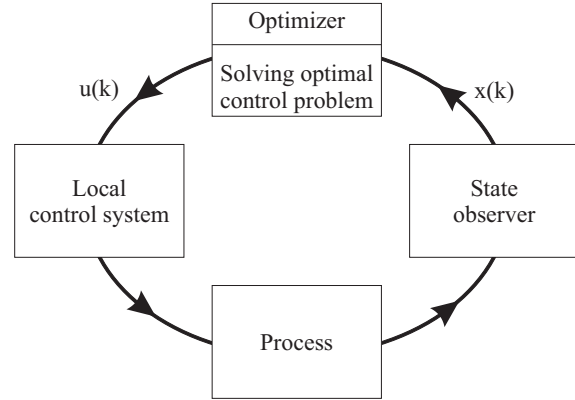


Figure 1. Nonlinear model predictive control (NMPC) scheme

Due to the combined character of the approach to be applied the model equations are not directly integrated in the NLP formulation (2) but solved to obtain the state variables $\mathbf{x}_{p,i+1}$ by a collocation scheme. For detailed description of the NOCP and its transformation to an NLP refer to (Tamimi and Li, 2010, 2009; Lazutkin et al., 2014).

3 Solution method

3.1 Solution of the resulting NLP problem

For simplicity reasons the NLP problem (2) is rewritten in the compact form

$$\begin{aligned} \min_{\boldsymbol{\omega}} & \{J(\boldsymbol{\omega})\} \\ \text{s. t. } & \mathbf{E}(\boldsymbol{\omega}) = \mathbf{0}, \\ & \mathbf{S}(\boldsymbol{\omega}) \leq \mathbf{0}, \end{aligned} \quad (3)$$

where $\boldsymbol{\omega}$ contains all optimization variables, \mathbf{E} all equality, and \mathbf{S} all inequality constraints.

The NLP (4) can be solved using an interior-point optimization solver (e. g. Ipopt (Wächter and Biegler, 2006)). Hence, the barrier function formulation of the NLP reads

$$\begin{aligned} \min_{\boldsymbol{\omega}, \mathbf{z}} & \left\{ J(\boldsymbol{\omega}) - \mu \sum_{j=1}^{n_S} \ln(z_j) \right\} \\ \text{s. t. } & \mathbf{E}(\boldsymbol{\omega}) = \mathbf{0}, \\ & \mathbf{S}(\boldsymbol{\omega}) + \mathbf{z} = \mathbf{0}, \end{aligned} \quad (4)$$

with a slack variable \mathbf{z} and the corresponding Lagrange function

$$\begin{aligned} \mathcal{L}(\boldsymbol{\omega}, \mathbf{z}, \boldsymbol{\lambda}) & = J(\boldsymbol{\omega}) - \mu \sum_{j=1}^{n_S} \ln(z_j) + (\boldsymbol{\lambda}^E)^T \mathbf{E}(\boldsymbol{\omega}) \\ & \quad + (\boldsymbol{\lambda}^S)^T (\mathbf{S}(\boldsymbol{\omega}) + \mathbf{z}) \end{aligned} \quad (5)$$

for a fixed value of the barrier parameter μ . The vector $\boldsymbol{\lambda}^T = [(\boldsymbol{\lambda}^E)^T, (\boldsymbol{\lambda}^S)^T] \in \mathbb{R}^{n_E+n_S}$ contains multipliers associated with the $n_E + n_S$ equality constraints. The iteration scheme of the interior-point algorithm is

$$\begin{aligned}\boldsymbol{\omega}_{l+1} &= \boldsymbol{\omega}_l + \alpha_l \cdot \Delta \boldsymbol{\omega}_l, \quad \mathbf{z}_{l+1} = \boldsymbol{\omega}_l + \alpha_l \cdot \Delta \mathbf{z}_l, \\ \boldsymbol{\lambda}_{l+1} &= \boldsymbol{\lambda}_l + \alpha_l \cdot \Delta \boldsymbol{\lambda}_l, \quad l = 0, 1, \dots\end{aligned}\quad (6)$$

The different search directions $\Delta \boldsymbol{\omega}_l$, $\Delta \mathbf{z}_l$, $\Delta \boldsymbol{\lambda}_l^E$, $\Delta \boldsymbol{\lambda}_l^S$ are obtained applying the Newton method to the KKT optimality conditions of problem (4). Let $\mathbf{Z} = \text{diag}(\mathbf{z})$ and $\mathbf{e}^T = (1, \dots, 1) \in \mathbb{R}^{n_S}$. Thus, the following system needs to be solved at each iteration step

$$\mathbf{K} \cdot \begin{bmatrix} \Delta \boldsymbol{\omega}_l \\ \Delta \mathbf{z}_l \\ \Delta \boldsymbol{\lambda}_l^E \\ \Delta \boldsymbol{\lambda}_l^S \end{bmatrix} = - \begin{bmatrix} \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}_l, \mathbf{z}_l, \boldsymbol{\lambda}_l) \\ -\mu \mathbf{e}^T \mathbf{Z}_l^{-1} + \boldsymbol{\lambda}_l^S \\ \mathbf{E}(\boldsymbol{\omega}_l) \\ \mathbf{S}(\boldsymbol{\omega}_l) + \mathbf{z}_l \end{bmatrix}, \quad \text{where} \quad (7)$$

$$\mathbf{K} = \begin{bmatrix} \nabla_{\boldsymbol{\omega}\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}_l, \mathbf{z}_l, \boldsymbol{\lambda}_l) & \mathbf{0} & \nabla \mathbf{E}(\boldsymbol{\omega}_l) & \nabla \mathbf{S}(\boldsymbol{\omega}_l) \\ \mathbf{0} & -\mu \mathbf{Z}_l^{-2} & \mathbf{0} & \mathbf{I}_{n_S} \\ \nabla \mathbf{E}(\boldsymbol{\omega}_l)^T & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \nabla \mathbf{S}(\boldsymbol{\omega}_l)^T & \mathbf{I}_{n_S} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

According to (7) the gradient $\nabla J(\boldsymbol{\omega}_l)$, Jacobians $\nabla \mathbf{E}(\boldsymbol{\omega}_l)$, $\nabla \mathbf{S}(\boldsymbol{\omega}_l)$, and Hessian matrices $\nabla^2 J(\boldsymbol{\omega}_l)$, $\nabla^2 \mathbf{E}_i(\boldsymbol{\omega}_l)$, $i = 1, \dots, n_E$, $\nabla^2 \mathbf{S}_j(\boldsymbol{\omega}_l)$, $j = 1, \dots, n_S$ are required and made available either analytically or approximately to the optimization solver. In the subsequent discussions the expression

$$\begin{aligned}\mathbf{H} &= \nabla_{\boldsymbol{\omega}\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}_l, \mathbf{z}_l, \boldsymbol{\lambda}_l) \\ &= \nabla^2 J(\boldsymbol{\omega}) + \sum_{i=1}^{n_E} \boldsymbol{\lambda}_i^E \nabla^2 \mathbf{E}_i(\boldsymbol{\omega}) + \sum_{j=1}^{n_S} \boldsymbol{\lambda}_j^S \nabla^2 \mathbf{S}_j(\boldsymbol{\omega})\end{aligned}\quad (8)$$

will be referred as the analytic Hessian (AH).

3.2 First- and second-order sensitivities

A collocation method is used in each shooting interval within the framework of the MCMSC approach. The states are approximated inside each shooting interval by a linear combination of the Lagrange polynomials (9) using a shifted Legendre collocation scheme,

$$\hat{x}(t) = \sum_{j=1}^{n_c} \left(\prod_{k=1, k \neq j}^{n_c} \frac{t - t_k}{t_j - t_k} \right) \cdot x_j^c, \quad (9)$$

with $\hat{x}(t)$ - a polynomial approximation of a single state variable, x_j^c - the unknown collocation-coefficient at the j -th collocation point, $\{t_1, \dots, t_{n_c}\}$ - collocation points in the shooting interval $[t_q, t_{q+1}]$. A shifted scheme means, that the last collocation point t_{n_c} is shifted to the right interval border - a necessity for continuity reasons - and the other ones are also shifted accordingly.

The derivative of the collocation polynomial reads

$$\frac{d\hat{x}(t)}{dt} = \sum_{j=1}^{n_c} \left(\frac{dl_j(t)}{dt} \right) x_j^c \quad (10)$$

$$\text{with } l_j(t) = \prod_{k=1, k \neq j}^{n_c} \frac{t - t_k}{t_j - t_k}$$

The parametrized states can be put into a vector $\mathbf{X}^{p,q}$ and the controls into a vector \mathbf{V}^q , where $q = 1, 2, \dots, N$. The components of the vectors $\mathbf{X}^{p,q}$ and \mathbf{V}^q are included in the decision variables \mathbf{X}^p and \mathbf{V} , respectively, in the NLP formulation. Furthermore, the vector $\mathbf{X}^{c,q}$ represents all collocation coefficients in the shooting interval q . Hence, the discretized nonlinear differential equation system results in the nonlinear algebraic equation system

$$\begin{aligned}\mathbf{G}^q &= \dot{\mathbf{W}} \cdot \mathbf{X}^{c,q} + \dot{\mathbf{W}}_0 \cdot \mathbf{X}^{p,q} \\ &\quad - \frac{(t_f - t_0)}{N} \cdot \mathbf{F}(\mathbf{X}^{c,q}, \mathbf{V}^q) = \mathbf{0}\end{aligned}\quad (11)$$

with $q = 1, 2, \dots, N$, q - index of shooting interval, N - number of shooting intervals, \mathbf{F} - discretized \mathbf{f} , $\dot{\mathbf{W}}$ and $\dot{\mathbf{W}}_0$ - derivative matrices of the Lagrange polynomials.

At each iteration step of the optimization procedure, for given values $\mathbf{X}^{p,q}$ and \mathbf{V}^q , (11) is solved by a Newton method. The results will be $\mathbf{X}^{c,q}$ as well as the first- and second-order sensitivities. These results are utilized in the SQP solver for calculation of the functions \mathbf{E} , \mathbf{S} , the Jacobians $\nabla \mathbf{E}$, $\nabla \mathbf{S}$, and the Hessian \mathbf{H} .

Neglecting the shooting interval index q and writing (11) in a compressed form delivers

$$\mathbf{G}(\mathbf{X}^c(\mathbf{X}^p, \mathbf{V}), \mathbf{X}^p, \mathbf{V}) = \mathbf{0}. \quad (12)$$

To obtain the first-order sensitivities, Eq. (12) has implicitly to be differentiated and provides

$$\frac{\partial \mathbf{G}}{\partial \mathbf{X}^c} \frac{\partial \mathbf{X}^c}{\partial [\mathbf{X}^{pT} \mathbf{V}^T]^T} - \frac{\partial \mathbf{G}}{\partial [\mathbf{X}^{pT} \mathbf{V}^T]^T} = \mathbf{0}. \quad (13)$$

Eq. (13) represents a linear equation system. Typically, $\frac{\partial \mathbf{G}}{\partial \mathbf{X}^c}$ and $\frac{\partial \mathbf{G}}{\partial [\mathbf{X}^{pT} \mathbf{V}^T]^T}$ have sparsity structures that can be exploited in determination of $\frac{\partial \mathbf{X}^c}{\partial [\mathbf{X}^{pT} \mathbf{V}^T]^T}$.

Using (13), analytic expressions are derived in order to calculate the second-order sensitivities. Hence, this equation is re-written here in the following compact form

$$\Phi(\mathbf{X}^c(\mathbf{X}^p, \mathbf{V}), \mathbf{X}^p, \mathbf{V}, \frac{\partial \mathbf{X}^c}{\partial [\mathbf{X}^{pT} \mathbf{V}^T]^T}(\mathbf{X}^p, \mathbf{V})) = \mathbf{0}. \quad (14)$$

The derivative $\frac{\partial \mathbf{X}^c}{\partial [\mathbf{X}^{pT} \mathbf{V}^T]^T}$ indicates the dependencies of the first-order sensitivities on the decision variables \mathbf{X}^p and \mathbf{V} . Applying the differentiation operator $\frac{\partial}{\partial [\mathbf{X}^{pT} \mathbf{V}^T]^T}$ to (14) the equations for second-order sensitivities will be available in matrix form and can be computed using LU decomposition.

4 Toolchain

As mentioned above in order to offer a comfortable and user-friendly way of modeling and optimizing physical systems, to unburden the user from automatable tasks like gradients and sensitivities calculations, to accelerate the implementation time as well as the computation time for solving the optimization problem, the implementation of the MCMSC method within an open-source toolchain is proposed. This toolchain consists of amongst others physically-based object-oriented modeling, using the modeling language Modelica with the extension Optimica, and the large scale nonlinear optimization solver Ipopt.

4.1 Components

Compared with block-oriented modeling, the object-oriented modeling approach provides a more comfortable and flexible alternative for physical systems. As a result this work uses the modeling language Modelica (Fritzon, 2014). Not only for simulation purposes but also for the formulation of different optimization tasks the platform JModelica.org is utilized. Besides the standard Modelica features JModelica.org contains the extension Optimica (Åkesson, 2008) including the possibility to formulate optimal control problems and other optimization tasks (Åkesson et al., 2010).

One essential tool utilized in many respects is the automatic differentiation tool CasADi (Andersson et al., 2011, 2012a,b). Hence, it is applied for the calculation of first- and second-order derivatives, symbolic manipulations of the objective and the constraints.

The standard multi-processing Python module (Hellmann, 2011) is a next module within the toolchain to perform parallel computations.

After adopting the optimization model to the MCMSC framework, Ipopt (Wächter and Biegler, 2006) is responsible for the solution of large scale nonlinear optimization problems. The interoperation of the toolchain within the optimizer (see Fig. 1) is illustrated in Fig. 2.

4.2 Functionality

After establishing an optimization model and implementing it by means of Modelica and the extension Optimica, the JModelica.org compiler transforms the model into a symbolic one. From the transformed model, i.e., model equations, variables, etc. are accessible by the Python scripting language.

The proposed MCMSC approach belongs to the category of quasi-sequential methods, i.e. the interior-point optimizer Ipopt solves in every iteration the state equations (11) and calculates the sensitivities (13) for given parametrized states and controls of each shooting interval. If the advantageous feature of an analytical Hessian symbolically calculated by CasADi is used, also in ev-

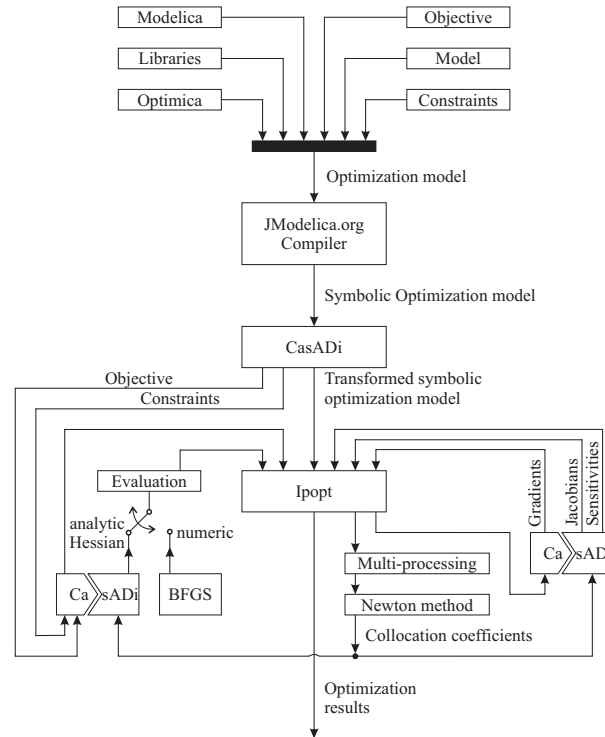


Figure 2. Parallelized MCMSC toolchain

ery iteration only the numerical values of the variables mattering have to be updated. It is also possible that numerical Hessians approximated through a BFGS formula which usually incurs more computational effort.

The MCMSC framework consists of three main parts, i.e. the optimizer, the calculation of state trajectories by means of a Newton method, and the sensitivity computations. Ideally, both the Newton method and the sensitivity calculation are recommended to be executed in parallel in each shooting interval. Depending on the computer architecture (multi-core, etc.) the user can define the number of processes. On the one hand, one gains computing time improvements via parallelization. On the other hand, the communication effort increases, the more parallel threads occur. There is a maximum speed up depending on the size of the optimization problem and the computer architecture.

Concerning the first-order sensitivities computations from (13), an LU decomposition and the direct solver for sparse matrices CSpase (Davis, 2006) are applied and interfaced to CasADi. The second-order derivatives, if used, are transformed to a linear equation system and also solved by CasADi. This symbolic tool is furthermore responsible for the generation of the Jacobians, the gradients of the objective function, and the symbolic manipulations of the objective functions and the constraints.

The entire approach of the parallelized MCMSC method is realized in the Python scripting language using standard multi-processing module without any additional software packages.

Table 1. Classes and functions

(C)lass/(F)unction name	Functionality
loading(object)	auxiliary functions to prepare the optimization problem for Ipopt
initialization	loads options and uses JModelica.org to extract the optimization problem from Modelica/Optimica.
extract	prepares the optimization problem
define_collocation	constructs Lagrange polynomials and derivative matrices
construct_vars	creates vectors of discretization
discretize	implicit DAE discretization
create_solvers	creates for each interval the Newton solver and solvers for first-order derivatives
interval_simulation	simulates the DAE system for given parameters
constr_vec	constraints vector for Ipopt
jacobian_of_constraint_vector	Jacobian for Ipopt
create_cost	objective for Ipopt
prep_ipopt	required information for Ipopt, e.g. number of non-zeros in Jacobians, boundaries
prep_sys_for_multiproc	divide the system according to the number of cores
optimize	solve the optimization problem by Ipopt
exact_hessian	construct exact Hessian and corresponding linear solvers
scaling	scale the problem

4.3 Source code examples

Different classes and functions shown in Tab. 1 are realized dedicated to certain purposes. In particular, there are several important issues in the MCMSC toolchain. A brief description is given below with some source code fragments. Using JModelica.org the formulated optimization problem can be easily transferred for further manipulations using the function `transfer_optimization_problem`, which has two attributes: `name` is an optimization problem file and `file_path` is a path to this file. Let `OP` be the symbolic representation of the dynamic optimization problem, which includes all required information. To see the list of functions and methods for the `OP` variable, users have to refer to the JModelica.org source code files.

The proposed toolchain requires a lot of functions from CasADi. For simplification, functionality of this software will be made completely available in the toolchain by importing all CasADi classes.

The first essential aspect is to get information about the declared variables (differential and algebraic states, controls, parameters) in the optimization problem formulation. The following has been implemented by the developer:

```
# Differential states
DIFF = OP.getVariables( ...
    OP.DIFFERENTIATED)
# Algebraic states
ALG = OP.getVariables( ...
    OP.REAL_ALGEBRAIC)
# Derivatives
```

```
DER = OP.getVariables(OP.DERIVATIVE)
# Controls
INPUT = OP.getVariables(OP.REAL_INPUT)
# Parameters
P_I = OP.getVariables( ...
    OP.REAL_PARAMETER_INDEPENDENT)
P_D = OP.getVariables( ...
    OP.REAL_PARAMETER_DEPENDENT)
```

In order to extract model dynamics, the `OP` variable has specific function to get DAEs, which returns a residual between left and right hand sides of the equations.

```
DAE = OP.getDaeResidual()
```

For further manipulations with the extracted DAE, the symbolic function using CasADi has to be established. To achieve this goal, all variables should be aggregated into an input vector.

```
MX_DAE = MXFunction(LIST_DER + ...
    LIST_DIFF + LIST_ALG + ...
    LIST_INPUT + P_I + P_D, [DAE])
MX_DAE.init()
```

Since JModelica.org works with the `MX` data type and the proposed toolchain accepts currently the `SX` data type, the `MXFunction` can be converted to `SXFunction`:

```
SX_DAE = SXFunction(MX_DAE)
SX_DAE.init()
```

For further information about data types in CasADi refer to the manual.

The function `SX_DAE` should be called with two arguments, the `SX` symbolic expression (converted from `MX`) and the scaled DAE system (with respect to the interval length).

```
SX_DAE_NUM = SX_DAE.call( ...
  SX_INPUTS_DAE)[0]
SX_DAE_FUNCTION = SXFunction( ...
  [vertcat(SX_INPUTS_DAE)], ...
  [LENGTH*SX_DAE_NUM])
SX_DAE_FUNCTION.init()
```

This function `SX_DAE_FUNCTION` is involved in the discretization procedure, since it declares the variable order and accepts symbolical evaluation.

For the discretization of the model equations, additional variables should be introduced,

```
# Piecewise control
CTRL = SX.sym("c", N_INT*N_C)
# Parameterized states
P_S_P = SX.sym("ps", (N_INT+1)*(N_D))
# Collocated differential
# and algebraic states
S_P = SX.sym("s", ...
  N_INT*((N_D + N_A)*NCP))
```

where `N_INT` is the number of shooting intervals defined by the user, `N_D`, `N_A`, and `N_C` are the numbers of differential, algebraic, and control variables.

For each shooting interval, certain variables are chosen from vectors `CTRL`, `P_S_P`, and `S_P`. The `SX_DAE_FUNCTION` is called with these variables and the evaluation results are placed into `RES` variable. This procedure should be called for each interval.

```
RES = SX_DAE_FUNCTION.call( ...
  [vertcat([der, diff, alg, ctrl, ...
  p_i_v, p_d_v])])[0]
```

As mentioned before, the state trajectories and sensitivities have to be calculated for each interval. For this purpose, the toolchain uses Newton and LU solvers from CasADi.

```
# Newton solver
# interval_dae - discretized DAE
# system for one interval
# variables - states at collocation
# points
# parameters - parametrized states
# and controls
F = SXFunction( ...
  [vertcat([variables]), ...
  vertcat([parameters])], ...
  [vertcat([interval_dae])])
solver = ImplicitFunction("newton", F)
solver.setOption("linear_solver", ...
```

```
"csparse")
solver.setOption("abstol", 1e-12)
solver.init()
# LU solver
# NCP - number of collocation points
SYSTEM_INDEX = (N_D + N_A)*NCP
# Full Jacobian
partial_jacobian=interval_dae.jac()
dGdX_sym = partial_jacobian[ ...
  range(SYSTEM_INDEX), ...
  range(SYSTEM_INDEX)]
LHS = MX.sym('LHS', ...
  dGdX_sym.sparsity())
dGdXp_dU_sym = partial_jacobian[ ...
  range(SYSTEM_INDEX), ...
  SYSTEM_INDEX:SYSTEM_INDEX+N_D+N_C]
RHS = MX.sym('RHS', ...
  dGdXp_dU_sym.sparsity())
LUSolver = solve(LHS, RHS, "csparse")
FRHS = MXFunction( ...
  [LHS, RHS], [LUSolver])
FRHS.init()
```

The optimization constraints vector and the corresponding Jacobian matrix, objective function and its gradient are also constructed by means of CasADi using symbolic manipulation. For the sake of brevity, these details are not discussed here.

Corresponding to the number of the user-defined processes in the case of a multi-core CPU, shooting intervals can be equally distributed between them.

After problem initialization, the `Ipopt` instance is created to solve the optimization problem:

```
import pyipopt
nlp = pyipopt.create(args)
x_opt = nlp.solve(initialGuess)
```

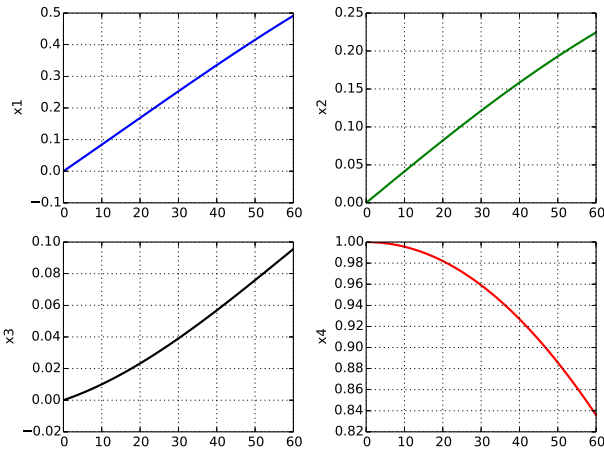
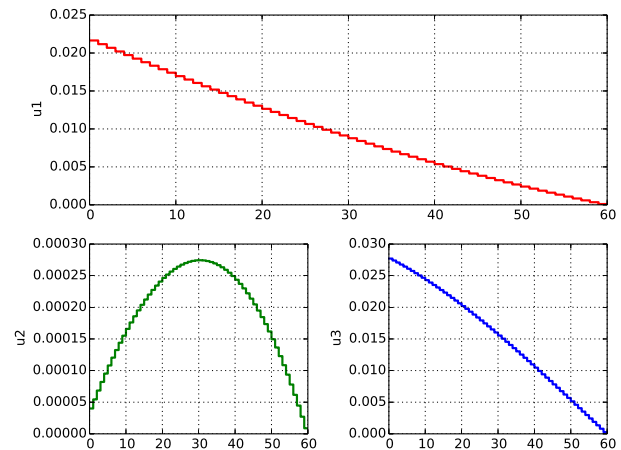
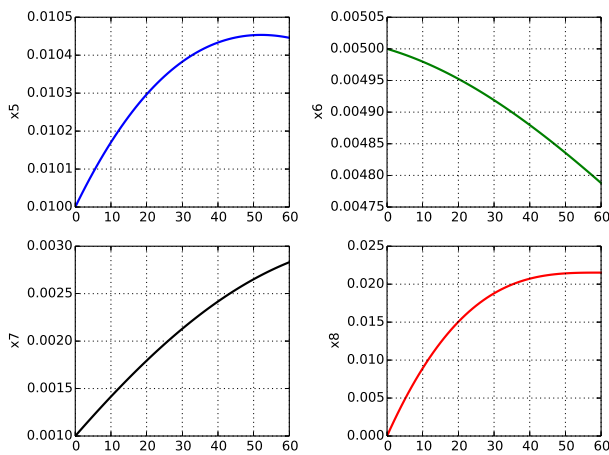
5 Examples

Showing the efficiency of the proposed approach a small-scale and a large-scale problem are presented. All computations are performed on a stand-alone personal computer with Intel[®] I7 4.4 GHz, 6 cores, 16 GB RAM with Ubuntu 14.04.1 Server x64 operational system.

5.1 Satellite control problem

This nonlinear optimal control problem is devoted to the calculation of the optimal control and the concluding optimal torques according to a Bolza functional that bring the satellite to rest after an initial tumbling motion. The problem is listed e.g. in (Rudquist and Edvall, 2009).

The optimal states are shown in Figs. 3 and 4, where the state x_8 represents the integrand of the Lagrange term. The optimal controls are contained in Fig. 5. The time horizon corresponds to 100 seconds and is divided


Figure 3. Optimal states $x_1(t)$ to $x_4(t)$

Figure 5. Optimal controls $u_1(t)$ to $u_3(t)$

Figure 4. Optimal states $x_5(t)$ to $x_8(t)$
Table 2. Satellite control problem dimensions

Number of . . .	Value
intervals	60
non-zeros in Jacobians of eqs.	5,768
non-zeros in Jacobians of ineqs.	0
non-zeros in Hessian of Lagrangian	3,960
equality constraints	480
inequality constraints	0
variables	668
variables incl. Newton variables	2,108

into 60 intervals. Figs. 3 - 5 show the correct results. The optimal objective value is in every scenario $J^* = 0.4639$.

Data being constant through all scenarios are listed in Tab. 2. In case of the utilization of the analytical Hessians the number of non-zero elements in the Hessian of the Lagrangian equals to 3,960.

Tab. 3 shows the speed-up in different scenarios. In this small-scale problem the effect of parallelization is not substantial, but the number of iterations can be reduced and thus the computation time is less in most cases.

In a first case, comparing the speed-up by *parallelization within the same computation scheme for the Hessian* (column s_1), i. e. considering the first and the second row in the column s_1 in BFGS, AH, and AHC case, resp., the speed-up factors s_1 reach from 1.08 (AHC), 1.33 (BFGS) to 1.56 (AH).

In the second case, comparisons are dedicated to the *non-parallelized versions* (column s_2) contrasting the Hessian calculation methods to each other. Unsurprisingly, no speed-up is achieved ($AH/n_c = 1/s_2 = 0.71$) due to the fact that the symbolic calculations take time to es-

Table 3. Speed-up by parallelization and utilization of analytic Hessian

Hess.	n_c	It.	t_Σ [s]	s_1	s_2	s_3
BFGS	1	8	0.515	1.00	1.00	N/A
	6	8	0.355	1.33	N/A	1.00
AH	1	5	0.723	1.00	0.71	N/A
	6	5	0.462	1.56	N/A	0.77
AHC	1	5	0.346	1.00	1.49	N/A
	6	5	0.320	1.08	N/A	1.11

Hess.: BFGS - approximated Hessian, AH - analytic Hessian (updated in every iteration of the optimizer), AHC - analytic Hessian (calculated once in iteration 0); n_c - no. of CPU cores ($n_c = 1$: no parallelization), It. - no. of iterations, t_Σ - total CPU time (mean value of 100 runs), s_i , $i = 1, 2, 3$ - speed-up factors, N/A - not applicable

establish the analytic Hessian. But, calculating the Hessian only once in the start iteration and leaving it constant through the iteration process also delivers the correct result with the risk that search direction could be non-descent, and the speed-up amounts to 1.49 ($AHC/n_c = 1/s_2 = 1.49$).

The third case considers the *parallelized versions* (column s_3), also contrasting the Hessian calculation methods to each other. The computation with the BFGS approximation constitutes the reference. As to be expected in this small-scale example, one gets also no speed-up in the parallelized versions ($AH/n_c=4/s_3 = 0.77$) from BFGS approximation to analytic Hessian calculation. However, confronting BFGS with AHC, the speed-up accounts to 1.11 ($AHC/n_c = 4/s_3 = 1.11$).

5.2 Combined cycle power plant start-up control problem

Another example, a combined cycle power plant (CCPP) was chosen for several reasons. Firstly, this is an example of interest in the liberalized energy market, because classical power plants have to be adopted to the operation of electrical energy supply networks with renewable energies. Thus, they are more often set into operation or shutdown than in the past. Secondly, it is a high-dimensional problem compared with other academic examples. Thirdly, the example is used for the verification of the achieved results. In (Casella and Pretolani, 2006) the system was introduced. The plant is composed of a gas turbine unit, heat recovery steam generator, a steam turbine, and a condenser. The start-up time is limited due to the following facts: a maximum load change rate of the gas turbine, the thermal stress in the thick components (e.g. steam turbine shafts), and limited control variables.

Several authors used the object-oriented implemented model for the optimal control of the start-up process. In (Casella et al., 2011a) a simplified model is used. The contribution (Casella et al., 2011b) reports on a solution using JModelica.org, CppAD for automatic differentiation, and Ipopt for the NLP solution. The integration of CasADi and JModelica.org is described in (Andersson et al., 2011), where the CCPP system is also used as a benchmark system, but the solution is achieved by a direct collocation approach. An approach with OpenModelica and an optimization language specification, CasADi as the automatic differentiation tool, and different optimization methods including direct collocation and direct multiple shooting, is shown in (Shitahun et al., 2013). A parallel multiple shooting and a collocation optimization, performed with OpenModelica, is explained in (Bachmann et al., 2012). That paper discusses multiple shooting, multiple collocation, and total collocation methods using up to 8 cores of a multi-core CPU with OpenMP support. In (Ruge et al., 2014) the authors outline a toolchain including modeling with OpenModelica,

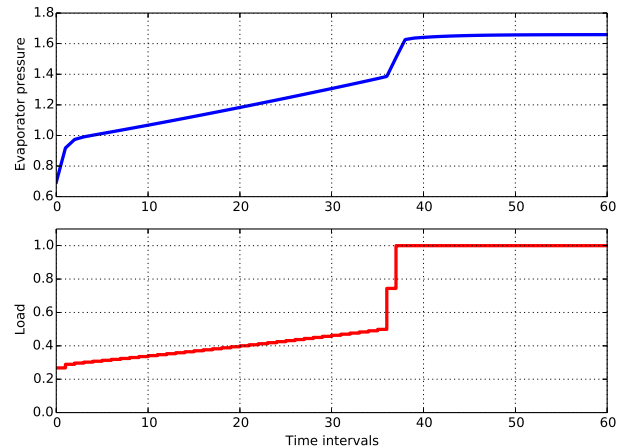


Figure 6. Optimal pressure and load

Table 4. CCPP problem dimensions

Number of ...	Value
intervals	60
non-zeros in Jacobians of eqs.	7,210
non-zeros in Jacobians of ineqs.	104,940
non-zeros in Hessian of Lagrangian	3,960
equality constraints	610
inequality constraints	9,540
variables	670
variables incl. Newton variables	24,790

but using automatic differentiation by ADOL-C.

The approach and toolchain discussed in our paper uses a modified combined multiple shooting and collocation (MCMSC) method, CasADi for automatic differentiation, JModelica.org for modeling and formulation of the optimization problem by means of Optimica, and Ipopt as NLP solver. Thus, a direct comparison to the contributions mentioned above is not possible due to different models, approaches, time horizons, tools, and computers used. Therefore, the only direct comparison between MCMSC and collocation method on finite elements using JModelica.org is given in Tab. 5.

Exemplarily, one of the essential optimal states (evaporator pressure) and the control (normalized load) are shown in Fig. 6 above and below, resp., over 60 time intervals corresponding to 4,000 seconds operation time, indicating the right behavior.

Data being constant through all scenarios are listed in Tab. 4. Using the analytical Hessians the number of non-zero elements in the Hessian of the Lagrangian equals to 3,960. Tab. 5 shows the acceleration of computation time in most cases.

Concerning this large-scale problem, the gain achieved by *parallelization within the same computation method for the Hessian* (column s_1) is better than in the small-scale problem above. Comparing the first and the second row in the column s_1 in each case (BFGS, AH,

Table 5. Speed-up by parallelization and utilization of analytic Hessian – comparison with collocation approach

MCMSC approach							
Hess.	n_c	It.	t_O [s]	t_{SN} [s]	s_1	s_2	s_3
BFGS	1	47	5.578	8.887	1.00	1.00	N/A
	6	47	5.585	4.509	1.97	N/A	1.00
AH	1	35	2.032	27.612	1.00	0.32	N/A
	6	35	2.038	12.530	2.20	N/A	0.36
AHC	1	34	1.044	7.112	1.00	1.25	N/A
	6	34	1.066	5.114	1.39	N/A	0.88
Collocation approach							
Hess.	n_c	It.	t_C [s]	s_{CMP}			
BFGS	1	54	11.446	1.13			
AH	1	60	7.892	0.54			
AHC	1	77	6.299	1.02			

Hess.: BFGS - approximated Hessian, AH - analytic Hessian (updated in every iteration), AHC - analytic Hessian (calculated once in iteration 0); n_c - no. of CPU cores, It. - no. of iterations, t_O - CPU time for optimization by Ipopt (not parallelizable), t_{SN} - CPU time for sensitivity calculation and Newton solver, t_C - CPU time for pure collocation on finite elements; (all CPU times are averages of 100 runs), s_i , $i = 1, 2, 3$ - speed-up factors, s_{CMP} - speed-up factor between collocation (CM) and parallelized MCMSC method, N/A - not applicable

and AHC, resp.), the speed-up factors s_1 reach from 1.39 (AHC), 1.97 (BFGS) to 2.20 (AH) referred to t_{SN} .

In the second case, the *non-parallelized versions* (column s_2) are under consideration referring the Hessian calculation methods to each other. Here, a speed-up is only achieved in the AHC case ($AHC/n_c = 1/s_2 = 1.25$).

The third comparison evaluates the *parallelized versions* (column s_3), again contrasting the Hessian calculation methods to each other. Here, no speed-ups are achieved. Nevertheless, considering the optimization time t_O in the AH and AHC case compared with the BFGS case, it is significantly reduced by factors of 2.74 and 5.24, resp., because of conducive matrix structures.

To have at least one comparison on the same computer of the MCMSC method with collocation method (CM) on finite elements used in JModelica.org the lower part in Tab. 5 was added. In the parallelized version of the MCMSC method both the BFGS ($BFGS/n_c = 6/t_O + t_{SN} = 10.094$) and the AHC scenario ($AHC/n_c = 6/t_O + t_{SN} = 6.180$) are faster than the non-parallelized version of the CM.

The investigations and presented results show that the presented parallelized MCMSC approach is a powerful solution technique solving optimal control problems within the proposed toolchain. The number of iterations are reduced compared with both the non-parallelized cases and the collocation approach, but the effort in one

iteration is typically higher if the analytic Hessian is used. The approach can most advantageously be applied to large-scale optimization problems.

6 Summary and Conclusions

An optimal control problem needs to be solved online in NMPC. This poses a challenge in the implementation of the numerical algorithms and the enhancement of the computation efficiency for the dynamic optimization approach. In this work, a toolchain for solving nonlinear dynamic optimization problems is developed based on the combined multiple shooting and collocation method. The toolchain is implemented in open-source software and both the first and the second-order sensitivities are automatically computed. As a result, the user needs only to provide the defined optimal control problem for implementing NMPC. In addition, parallel computing is realized for performing the computations in the individual time intervals, thus leading to a reasonable reduction of the computation time. The results of two case studies show the capability of the toolchain for efficiently solving small to large-scale dynamic optimization problems.

In future, it is planned to offer a web-based optimization service for solving nonlinear dynamic optimization problems using the proposed approach.

7 Acknowledgments

The authors are grateful for the support of the ITEA2/EUREKA Cluster programme by the European Commission (project no. 11004, Model Driven Physical Systems Operation (MODRIO)) and for the financing by German Ministry of Education and Research (BMBF, Förderkennzeichen: 01IS12022H).

References

- J. Åkesson. Optimica – an extension of Modelica supporting dynamic optimization. In *Proc. 6th Int. Modelica Conf.*, pages 57–66. Modelica Association, March 3-4 2008.
- J. Åkesson, K.-E. Årzén, M. Gåfvert, T. Bergdahl, and H. Tummescheit. Modeling and optimization with Optimica and JModelica.org-languages and tools for solving large-scale dynamic optimization problems. *Comput. Chem. Eng.*, 34(11):1737–1749, 2010.
- J. Andersson, J. Åkesson, F. Casella, and M. Diehl. Integration of CasADi and JModelica.org. In *Proc. 8th Int. Modelica Conf.*, pages 218–231, 2011. doi:10.3384/ecp11063.
- J. Andersson, J. Åkesson, and M. Diehl. Dynamic optimization with CasADi. In *51st IEEE Conference on Decision and Control*, pages 681–686, 10-13 December 2012a. doi:10.1109/CDC.2012.6426534.

- J. Andersson, J. Åkesson, and M. Diehl. CasADi: A symbolic package for automatic differentiation and optimal control. *Lect. Notes Comput. Sci. Eng.*, 87:297–307, 2012b.
- 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 *Proc. 9th Int. Modelica Conf.*, pages 659–668, 2012.
- E. Balsa-Canto, J. R. Banga, A. A. Alonso, and V. S. Vassiliadis. Efficient optimal control of bioprocesses using second-order information. *Ind. Eng. Chem. Res.*, 39(11):4287–4295, 2000. doi:10.1021/ie990658p.
- M. Bartl, P. Li, and L. T. Biegler. Improvement of state profile accuracy in nonlinear dynamic optimization with the quasi-sequential approach. *AIChE J.*, 57(8):2185–2197, 2011. doi:10.1002/aic.12437.
- T. Barz, R. Klaus, L. Zhu, G. Wozny, and H. Arellano-Garcia. Generation of discrete first- and second-order sensitivities for single shooting. *AIChE J.*, 58(10):3110–3122, 2012.
- L. T. Biegler, A. M. Cervantes, and A. Wächter. Advances in simultaneous strategies for dynamic process optimization. *Chem. Eng. Sci.*, 57(4):575–593, 2002.
- F. Casella and F. Pretolani. Fast Start-up of a Combined-Cycle Power Plant: a Simulation Study with Modelica. In *Proc. 5th Modelica Conf.*, pages 3–10, 2006.
- F. Casella, F. Donida, and J. Åkesson. Object-Oriented Modeling and Optimal Control: A Case Study in Power Plant Start-Up. In *Prepr. 18th IFAC World Congress. Milano. Italy*, pages 9545–9554, 2011a.
- F. Casella, M. Farina, F. Righetti, R. Scattolini, D. Faille, F. Davelaar, A. Tica, H. Gueguen, and D. Dumur. An optimization procedure of the start-up of Combined Cycle Power Plants. In *Prepr. 18th IFAC World Congress. Milano. Italy*, pages 7043–7048, 2011b.
- T. A. Davis. *Direct methods for sparse linear systems*. SIAM, 2006.
- P. Fritzson. *Principles of object-oriented modeling and simulation with Modelica 3.3: A cyber-physical approach*. Wiley-IEEE Press, 2014.
- D. Hellmann. *The Python standard library by example*. Addison-Wesley Professional, 1st edition edition, 2011.
- W. R. Hong, S. Q. Wang, P. Li, G. Wozny, and L. T. Biegler. A quasi-sequential approach to large-scale dynamic optimization problems. *AIChE J.*, 52(1):255–268, 2006. doi:10.1002/aic.10625.
- B. Houska, H. J. Ferreau, and M. Diehl. An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, 47(10):2279–2285, 2011.
- C. Kirches, L. Wirsching, H. G. Bock, and J. P. Schlöder. Efficient direct multiple shooting for nonlinear model predictive control on long horizons. *J. Process Contr.*, 22(3):540–551, 2012.
- E. Lazutkin, A. Geletu, S. Hopfgarten, and P. Li. Modified multiple shooting combined with collocation method in JModelica.org with symbolic calculations. In *Proc. 10th Int. Modelica Conf.*, pages 999–1006, 2014. doi:10.3384/ECP14096999.
- D. Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.
- P. E. Rudquist and M. M. Edvall. PROPT - Matlab Optimal Control Software. User’s Guide. TOMLAB Optimization, 2009.
- V. Ruge, W. Braun, B. Bachmann, A. Walther, and K. Kulshreshtha. Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL-C. In *Proc. 10th Int. Modelica Conf.*, pages 1017–1025, 2014. doi:10.3384/ECP140961017.
- A. Schäfer, P. Kühn, M. Diehl, J. Schlöder, and H. G. Bock. Fast reduced multiple-shooting method for nonlinear model predictive control. *Chem. Eng. Process.*, 46(11):1200–1214, 2007.
- 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 *7th IFAC Symp. on Advances in Automotive Control*, volume 1, pages 446–451, 2013. doi:10.3182/20130904-4-JP-2042.00166.
- J. Tamimi and P. Li. Nonlinear model predictive control using multiple shooting combined with collocation on finite elements. In *7th IFAC Int. Symp. on Advanced Control of Chemical Processes*, pages 703–708, 2009. doi:10.3182/20090712-4-TR-2008.00114.
- J. Tamimi and P. Li. A combined approach to nonlinear model predictive control of fast systems. *J. Process Contr.*, 20(9):1092–1102, 2010.
- A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.
- Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE T. Contr. Syst. T.*, 18(2):267–278, 2010.
- D. P. Word, J. Kang, J. Åkesson, and C. D. Laird. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Comput. Optim. Appl.*, 59(3):667–688, 2014. doi:10.1007/s10589-014-9651-2.
- V. M. Zavala, C. D. Laird, and L. T. Biegler. Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chem. Eng. Sci.*, 63(4834-4845):19, 2008.

NMPC Application using JModelica.org: Features and Performance

Christian Hartlep¹ Toivo Henningsson²

¹Siemens AG, Germany, christian.hartlep.ext@siemens.com

²Modelon AB, Sweden, toivo.henningsson@modelon.com

Abstract

In the past JModelica.org was successfully applied for generating optimal trajectories. Using it for Nonlinear Model Predictive Control (NMPC) is the natural next step and sets high requirements on calculation time. To improve real time capabilities warmstarting of the optimization and elimination of algebraic variables based on Block Lower Triangular (BLT) form were implemented. In performance comparisons, using the example of steam temperature control, a speed-up of the optimization time by a factor of five and of two respectively was measured. The increased efficiency allows application of NMPC to faster systems than before.

Keywords: NMPC, BLT, IPOPT, JModelica.org

1 Introduction

A complete computational framework for Nonlinear Model Predictive Control (NMPC) demands various features provided by the FMI standard and its tool implementation, here JModelica.org by Modelon AB. This includes FMI2.0 linearization and elimination of algebraic variables based on the Block Lower Triangular (BLT) form, which recently became available. In the field of optimization it is intended to apply Modelica tools to large systems with safety and real-time restrictions. Effective equation pre-processing and efficient solving is demanded. For the latter warmstarting of optimizations improved real-time capabilities significantly. In this article the performance impact of different features is analyzed using the example of steam temperature control.

In Section 2 the NMPC framework is described with a focus on how JModelica.org tools are used for it. Recent enhancements for this particular application are discussed theoretically in Section 3 and analyzed regarding performance in Section 4. Finally Section 5 gives a short summary.

2 NMPC Loop Implementation

JModelica.org offers a basic framework for NMPC, which is described in (Axelsson et al., 2015). Here it is not used due to specific requirements for the intended application. This section gives an overview of this NMPC framework and how JModelica.org tools are used in it.

2.1 General Framework

The NMPC framework consists mainly of an observer and an optimizer, which interact with the plant as shown in Figure 1. In the top part the observer is depicted, which simulates the observer model based on the corrected state \hat{x}_{k+1}^+ of the previous time step and the current boundary conditions p_k . It generates a predicted state \hat{x}_{k+1}^- and predicted plant output \hat{y}_{k+1} , which is given to the corrector. For Kalman filter based observers and Luenberger observer the filter gain calculation requires linearization. In dependence of the difference between predicted and actual plant output y_k^{plant} the corrected state \hat{x}_{k+1}^+ is generated and used as initial state for the NMPC optimization. The NMPC generates the discrete future control input trajectory $u_{k,traj}$. In a post-processing step a simulation is performed to obtain continuous time control inputs $u^{NMPC}(t)$ for the plant. This allows excluding components which are relevant for the plant internal controllers but not the optimization. For this paper the plant is replaced by a simulation of a more complex model compared to the optimization model.

2.2 Observer

It is the observer's task to provide state estimates based on the estimated state of the previous time step and the plant outputs. Extended Kalman Filter (EKF) and Luenberger observer require the linearized state space representation for the calculation of the filter gain and a simulation of the nonlinear system. An Unscented Kalman Filter requires only the latter. The specific implementation is described in detail in (Bonvini et al., 2012). Implementing the EKF and Luenberger became more efficient recently due to the broad availability of FMI2.0 with

its ability to provide directional derivatives. Compared to the previous JModelica.org Model Unit (JMU)-based implementation the observer could be implemented more efficiently because simulation and linearization are handled by the same object. Benefits are reduced memory consumption and no need for transferring the predicted system state between the two objects.

2.3 Optimizer

Finding the optimal control inputs is handled by the JModelica.org optimization tool chain, which interfaces CasADi (Andersson, 2013) and IPOPT (Wächter and Biegler, 2006). It imports optimization problems written in Modelica/Optimica and discretizes them from continuous time to discrete time using collocation (Magnusson and Åkesson, 2012). Automatic generation of Jacobian and Hessian of the optimization problem is included. Recently, elimination of algebraic variables based on the BLT form of the resulting optimization problem was implemented to improve optimization solution time. Furthermore time-critical reoptimizations became faster due to reuse of the collocation and solver object and more importantly warmstarting of the optimization by providing primal and dual variables of the previous NMPC iteration. Both features are thoroughly explained in the next section. Another beneficial optimization feature is custom distribution of mesh elements which allows appropriate sizing of the optimization problem. At the beginning of the time horizon the element size is mainly determined by the required update interval, whereas at the horizon end only numerical accuracy remains important. An example of mesh relocation is given in (Zhao and Tsiotras, 2011).

3 JModelica.org Enhancements to Better Support NMPC Applications

This section describes some recent enhancements to the optimization tool chain in JModelica.org that are useful to improve convergence speed and robustness in the context of NMPC.

3.1 Warm Starting

The aim of warm starting is to reduce solution time and improve convergence robustness for repeated solution of similar optimization problems, such as the ones found in e.g. NMPC. In JModelica.org, these effects are achieved by two means: reusing the discretization and reusing the latest solution to improve the initial guess. To explain what the former means, we first give a short background on the discretization scheme, collocation.

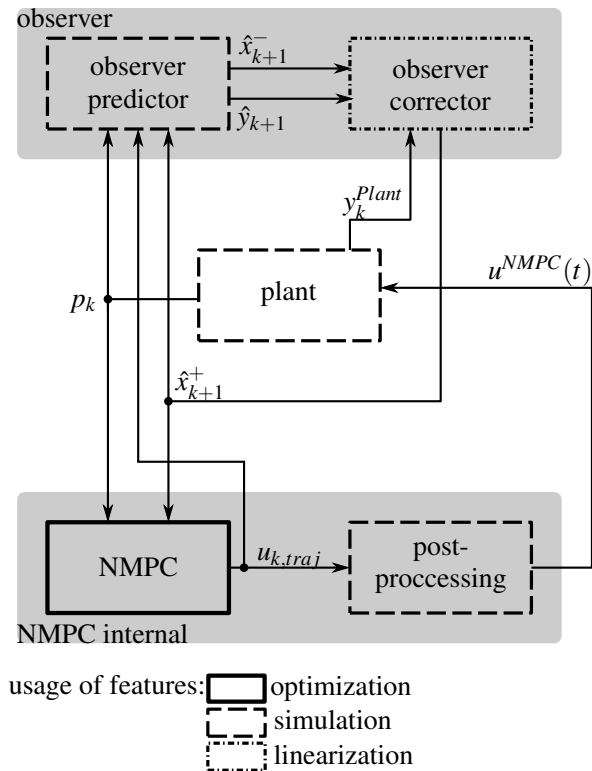


Figure 1. Structure of NMPC loop with signal exchanges and particular usage of JModelica.org feature

3.1.1 Collocation

When formulating an NMPC problem in Modelica/Optimica, modelling is naturally done in continuous time. To solve an optimization problem, we first *discretize* it by approximating it by a (large but sparse) Nonlinear Program (NLP) using collocation.

The time horizon is partitioned into *elements* (time intervals), and each time varying variable is approximated by a low order polynomial over each element. Each polynomial piece is described by sample values at a number of *collocation points* within the element.

3.1.2 Reusing the discretization

Creating the discretization can take a significant part of total solution time, sometimes even dominating it. To enable reuse, it must be valid not only for a single problem instance, but be parametrized by the degrees of freedom that can change from one time step to the next.

NLP parameters are introduced for

- Initial states.
- Parameters in the original model.
- External signals fed into the model, which can be used e.g. for time varying reference signals, or measurement signals in Moving Horizon Estimation.
- Scaling factors for equations.

The first three can be changed by the user between successive optimizations, while the last is typically determined automatically at the start of each optimization.

3.1.3 Reusing the latest solution

In NMPC, it is not only the optimization problems that are similar between time steps, but usually also their solutions. This feature can be exploited to minimize the amount of work for the nonlinear solver to find the solution to the next problem. We do this by

- Reusing primal and dual variables from the last solution to create the initial guess for the next. The primal variables represent the actual solution to the optimization problem, i.e. the trajectories of all variables, and can be shifted in time. If the last optimization failed, the solution from the last successful optimization can be used instead.

The dual variables represent the cost of violating the constraints that are active in the optimization problem (see e.g. (Boyd and Vandenberghe, 2004)). Primal-dual optimization algorithms such as IPOPT converge both the primal and dual variables simultaneously, so fast convergence requires a good initial guess for the dual variables as well. Since it is not obvious that shifting is applicable to dual variables, they are reused as is.

- Reusing the nonlinear solver state itself, which may contain factorizations etc. that might be reused from one time step to the next.

For IPOPT, which is an interior point solver, we also need to adjust the initial value of the barrier parameter μ so as not to push the initial guess too far away from the constraints.

3.2 Elimination of Algebraic Variables

Modelica models often contain very many algebraic variables. In simulation, values for all algebraic variables must be explicitly solved at each time step. In optimization based on collocation, on the other hand, it is possible to leave algebraic variables and the constraints that define them in the NLP, which will also contain constraints for the dynamics of the states.

Still, one can usually solve for some algebraic variables explicitly in terms of other, non-eliminated variables. Doing so brings a number of potential benefits for solving the NLP:

- Fewer decision variables to converge, to provide scalings and initial guesses for.
- Smaller matrices to factorize; factorization typically dominates the NLP solution time.

- Reduced sensitivity to the detailed formulation of the original Modelica model. Many formulations will yield equal NLPs, since intermediate variables introduced by the modeler can often be eliminated, producing the same results as if the intermediate was manually replaced by its definition.

Some variables might be left better uneliminated, however, if they can only be solved for iteratively or the elimination reduces sparsity too much in the NLP.

When importing a model for optimization in JModelica.org, an option has been introduced to eliminate algebraic variables. Suitable variables and equations are identified from the Block Lower Triangular (BLT) form of the model (the BLT form is constructed using Tarjan's Algorithm (Duff and Reid, 1978), starting from a perfect matching of the algebraic equation system).

The BLT form consists of an ordering of the equations and algebraic variables (non-states) in the algebraic equation system that must be solved to carry the solution of the system dynamics forward. It also contains a grouping of these equations and variables into diagonal blocks. To solve the entire equation system, each diagonal block can be solved in sequence, solving the system comprised of the block's equations for its unknowns. If a block requires the value of a variable outside of the block to solve it, that variable will already have been solved for in a previous block.

It is quite common for an algebraic variable to be expressed as an explicit function of one or more states and algebraic variables that can be computed before it. The relation between the algebraic variable and other variables will then be expressed by a scalar (1×1) BLT block with a single equation that the compiler can solve analytically; this is currently the only case used for elimination of algebraic variables. Elimination means that a variable will be calculated on the fly whenever needed, instead of requiring a nonlinear equation solver to iterate to find a solution for it. Variables with bounds are not eliminated since this will not reduce the size of the NLP.

As a simple example, consider the dynamic optimization problem

$$\begin{aligned} \min_{c,x,u} \int_{t=0}^1 c dt \\ \text{s.t. } c = x^2 + u^2, \quad \dot{x} = u, \quad x(0) = 1 \end{aligned}$$

where c , x , and u are to be determined as real valued functions of time. The algebraic equation system is comprised of the two constraints $c = x^2 + u^2$ and $\dot{x} = u$, with unknowns c and \dot{x} . The state derivative \dot{x} can only be solved from the latter equation, which leaves c to be solved from the former. This gives a BLT form with two scalar blocks.

The instantaneous cost c can be seen as an intermediate variable, and the BLT correctly identifies the opportunity to eliminate it, by matching it with the scalar

equation $c = x^2 + u^2$. Using this equation to explicitly solve for and eliminate the algebraic variable c results in the optimization problem

$$\begin{aligned} \min_{x,u} \int_{t=0}^1 x^2 + u^2 dt \\ \text{s.t. } \dot{x} = u, \quad x(0) = 1 \end{aligned}$$

which is smaller, with fewer equality constraints to satisfy and variables to converge.

In this case, the elimination of c yields additional benefits. The cost function in an optimization problem should be convex in a vicinity of the optimum (in this case, $x^2 + u^2$ is convex everywhere). Eliminating the intermediate variable c exposes this structure to the optimizer. In contrast, the original constraint $c = x^2 + u^2$ is non-convex even at the optimum, so keeping it will likely make it harder for the optimizer to find its way.

4 Performance Comparisons for Steam Temperature Control

In this section the performance impact of the discussed features is analyzed for the steam temperature control in a combined cycle power plant.

4.1 Test Setup

This section gives an overview of the optimization and plant models as well as the chosen scenario.

4.1.1 Soft- and Hardware

As software tools JModelica.org 1.16 (Magnusson and Åkesson, 2015), CasADi 1.9 with optimizer IPOPT 3.12 and HSL MA27 (HSL, 2013) as linear solver were used. The tests were carried out on a Intel i7 2620M (2.7 GHz) processor with Windows 7 x86.

4.1.2 Optimization Model

Figure 2 shows the optimization model with its three superheaters, four water injectors and two temperature sensors. Heat is transferred from flue gas (solid line) to two different steam lines (dash-dotted lines). The top and bottom superheater are part of the high pressure line (thickest dash-dotted line) and the middle one is part of the reheat line (thick dash-dotted line). It is the control target to provide a required steam temperature at the temperature sensors with minimal entropy production by changing the water valve openings. The optimization problem becomes nonlinear due to nonlinear water steam tables, nonlinear heat transfer functions in superheaters and the nonlinear objective function. In total the model includes six numerical states representing the physics, four control inputs and two outputs.

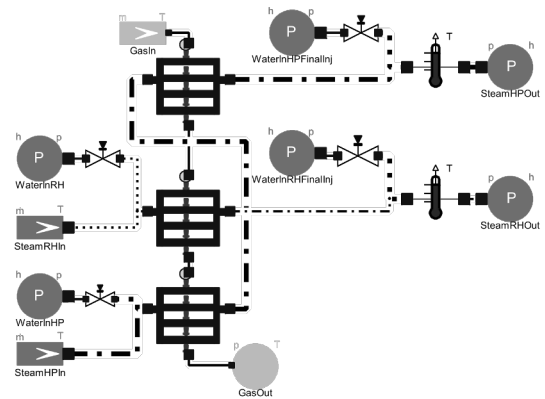


Figure 2. Optimization model for steam temperature control with superheaters, valves and temperature sensors

4.1.3 Plant Model

The performance of the NMPC loop is not tested on a real life plant but on a much more detailed model, which has been validated against measurement data of a real plant. This detailed plant model not only includes more heat exchangers than the model used for optimization (seven heat exchangers instead of three), but also intermediate piping between the heat exchangers as well as the underlying valve opening controllers for the injection valves. This results in a system of equations having about 14000 equations and 700 numeric states.

4.1.4 Test Scenario

As test scenario a plant startup is used because it includes various changes of setpoints and boundary conditions, which are not known in advance for the optimizer. Since the plant state changes drastically during this phase prediction accuracy is impaired especially at the beginning. The scenario is stopped before the plant reaches full load to capture only the dynamic behaviour and hence more challenging phases. Otherwise the results would be skewed in favour of the warmstart option.

In Figure 3 the resulting trajectories are plotted. The first two plots show measured, estimated and setpoint temperature for high pressure and reheat steam line respectively. The bottom subplot depicts the valve openings for intermediate (Int) and final injectors for each steam line. For the reheat part oscillations are visible, introduced by the stiff filter tuning. Such a filter tuning should not be used for a real plant but is beneficial for the performance test due to the stronger initial state update, which acts as disturbance for the optimizer.

4.2 Performance Comparison

For the application of NMPC low calculation time is required. Therefore features described in Section 3 are

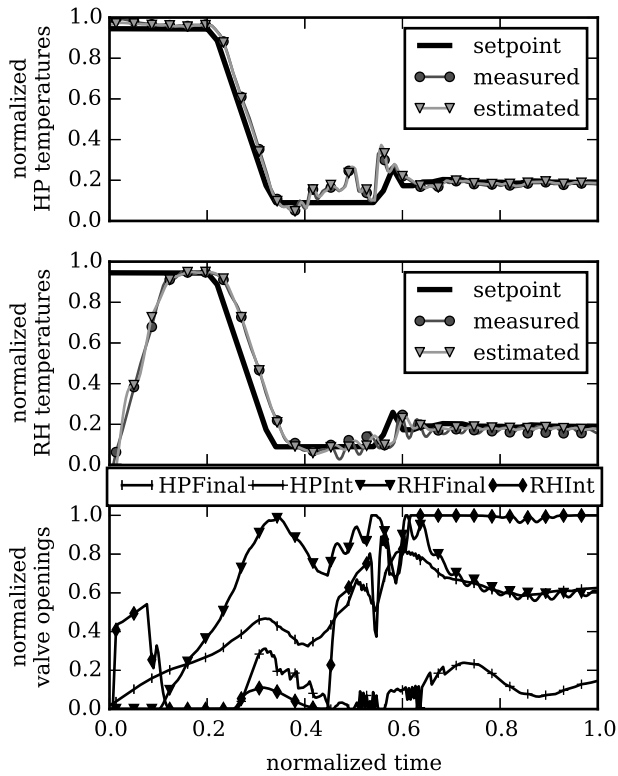


Figure 3. Test scenario for reheat (RH) and high pressure (HP) heat exchanger temperature control

tested here.

4.2.1 Criteria

Optimization performance is compared with regards to maximum and mean overall optimization time t_{opt} , maximum and mean IPOPT solution time t_{sol} , mean iterations n_{iter} , mean time per iteration. The overall optimization time includes setting initial trajectory and boundaries, JModelica.org preprocessing, optimization and JModelica.org post-processing. The gathered performance data is collected in Table 1.

4.2.2 Warmstart

Enabling the warmstart option improves performance in almost all performance criteria (see line 1 and 2 in Table 1) without changing the calculated trajectories. Warmstarting IPOPT reduced the average number of iterations and hence IPOPT solution time showed a reduction of 44%. Most notably is the reduction of the mean overall optimization time by 80% due to reusing of the solver object. An overview of the overall optimization time distribution is given in Figure 4. The distribution for warmstart is more narrow, which is visible in a reduction of the standard deviation from 0.4 s for the normal start to 0.25 s. Thus the optimization time became more predictable. Also the number of outliers is lower in the

warmstart distribution.

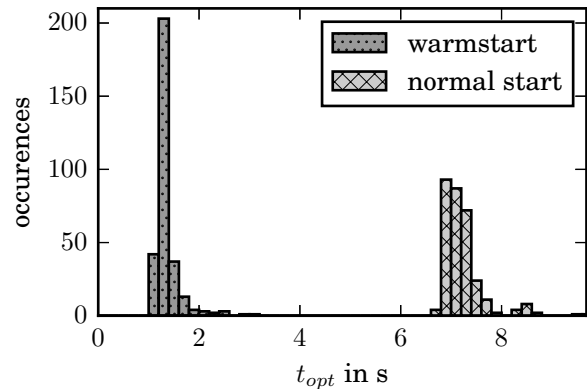


Figure 4. Comparison of total optimization time between normal optimization start and warmstart

4.2.3 BLT Elimination

Using BLT elimination improved performance and robustness in this case (see line 2 and 3 in Table 1). The most visible effect is the reduction of the number of variables by a factor of two (from 10359 to 5263, as reported by IPOPT). The smaller problem dimension translates into faster solution time and overall optimization time by about the same factor. The expected improvement in robustness was observed since 16 of 310 optimizations were unsuccessful with disabled BLT, whereas all optimizations were successful with enabled BLT. Higher time per iteration shows that the original problem formulation was suboptimal.

4.2.4 Scaling of Computational Cost

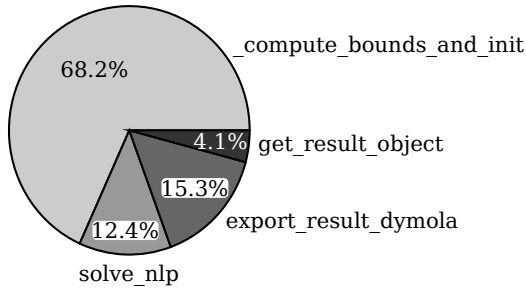
Scaling of computational cost is compared for different scopes of included components. The smaller comparison model includes only the last high pressure superheater of the high pressure line (the top one in Figure 2) and water injectors before and after. This system has two numerical states, two control inputs and one control output.

Number of variables in IPOPT is 3.6 times higher for the model with both steam lines compared to the version with only one steam line. For the tests BLT and warmstart are enabled and the results are summarized in line 4 of Table 1. The average overall optimization, average solution time and average time per iteration are 2.9, 3.6 and 2.4 times higher respectively for the larger model. These values are slightly lower than the increase in optimization problem size which means computational cost increases less than linear.

4.2.5 Code Analysis

In the last part of the analysis distribution of computation time to different subtasks, which are performed in

	Modelled Steam Lines	BLT	warmstart	$\max(t_{opt})$	\bar{t}_{opt}	$\max(t_{sol})$	\bar{t}_{sol}	\bar{n}_{iter}	$\bar{n}_{iter}/\bar{t}_{sol}$
1	Reheat and High Pressure	on	off	8.06 s	7.00 s	1.04 s	0.81 s	20.76	38.9 ms
2	Reheat and High Pressure	on	on	3.03 s	1.37 s	2.09 s	0.45 s	10.37	43.5 ms
3	Reheat and High Pressure	off	on	15.14 s	2.78 s	13.98 s	1.00 s	17.64	56.6 ms
4	only High Pressure	on	on	1.06 s	0.48 s	0.59 s	0.13 s	7.21	18.0 ms

Table 1. Performance comparison of different optimization setups

Figure 5. Percentage-wise breakdown of overall optimization time to different subtasks

every NMPC loop iteration, is analyzed. The result is visualized in Figure 5 for the computationally most expensive subtasks. All remaining subtasks take less than 2% together of the overall optimization time. Warmstart and BLT are enabled. About 32% of the overall optimization time is spent inside the optimize command of JModelica.org. Hence significant improvements in real time capabilities are possible going forward. Most time is spent for setting the initial trajectory (`_compute_bounds_and_init`). This step is not necessary for every NMPC loop iteration, but in case of an unsuccessful last optimization it is better to use the values of the last converged solution instead of the values that are still stored from the previous optimization. Therefore it is relevant for the analysis. Another time demanding task is exporting the result to a text file readable by Dymola. Whereas it is helpful for debugging the optimization result it could be omitted for time critical operations.

5 Summary

In conclusion the newly implemented features warmstart and BLT elimination in JModelica.org increased performance considerably for NMPC applications. JModelica.org with its Python interface proved to be viable choice for implementing an NMPC loop and was successfully tested for steam temperature control. Especially the warmstarting of the optimization improved tool capabilities in the field of NMPC application due to smaller optimization overhead and faster convergence. Variable elimination based on BLT reduced the optimization problem size and also gave a robustness improvement in the tests.

6 Acknowledgement

We would like to thank German Ministry BMBF for partially funding this work within the ITEA2 project MODRIO, as well as Siemens AG for providing further resources. For their critical responses and help we thank Prof. Pu Li, Kilian Link, Stephanie Gallardo Yances and Karin Dietl.

References

- HSL, a collection of fortran codes for large-scale scientific computation, 2013. URL <http://www.hsl.rl.ac.uk/>.
- Joel Andersson. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013.
- Magdalena Axelsson, Frederik Magnusson, and Toivo Henningsson. A framework for nonlinear model predictive control in jmodelica.org. *Proceedings of the 11th International ModelicaConference*, 2015.
- Marco Bonvini, Michael Wetter, and Michael D Sohn. An fmi-based framework for state and parameter estimation. *Proceedings of the 10th International ModelicaConference*, 2012.
- Stephen Boyd and Lieven Vandenbergh. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.
- I. S. Duff and J. K. Reid. An implementation of tarjan’s algorithm for the block triangularization of a matrix. *ACM Trans. Math. Softw.*, 4(2):137–147, June 1978. ISSN 0098-3500. doi:10.1145/355780.355785. URL <http://doi.acm.org/10.1145/355780.355785>.
- Fredrik Magnusson and Johan Åkesson. Collocation methods for optimization in a modelica environment. In *Proceedings of the 9th International Modelica Conference*, 2012.
- Fredrik Magnusson and Johan Åkesson. Dynamic optimization in jmodelica.org. *Processes*, 3(2):471, 2015. ISSN 2227-9717. doi:10.3390/pr3020471. URL <http://www.mdpi.com/2227-9717/3/2/471>.

Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. ISSN 0025-5610. doi:10.1007/s10107-004-0559-y. URL <http://dx.doi.org/10.1007/s10107-004-0559-y>.

Yiming Zhao and Panagiotis Tsiotras. Density functions for mesh refinement in numerical optimal control. *Journal of guidance, control, and dynamics*, 34(1):271–277, 2011.

A Modelica Library for Manual Tracking

James J. Potter

VTT Technical Research Centre of Finland
Vuorimiehentie 3, Espoo, Finland

Abstract

Many systems require a human to perform real-time control. To simulate these systems, a dynamic model of the human's control behavior is needed. The field of manual control has developed and validated such models, and their implementation in Modelica could support researchers of human-machine systems. This paper presents a Modelica library with models from the manual control literature. Python-based tools allow users to perform, in real time, the manual tracking tasks they design in Modelica. Parameter values in the manual controller models can be automatically tuned to either maximize tracking performance, or to match recorded control input from a user experiment.

Keywords: manual control, parameter estimation, FMI, Python, OpenModelica, JModelica.org

1 Introduction

There are many situations where a human operator attempts to make the output of a system follow a desired trajectory. For example, the top of Figure 1 shows the task of recording an athlete with a tripod-mounted video camera. The goal of the camera operator is to keep the athlete centered in the camera frame. The actual camera direction is compared to its desired direction (pointed directly at the athlete), and corrective actions are made by applying force to the tripod handle. This activity is similar to eye tracking, where a human keeps a moving object in the center of his or her vision (Jagacinski, 1977). In these activities, the human is an active part of a feedback control system. Other examples of manual tracking tasks include aiming a tank turret (Tustin, 1947; Kleinman and Perkins, 1974), driving an automobile (Bekey et al., 1977; Hess and Modjtahedzadeh, 1990), and piloting an aircraft (McRuer and Jex, 1967).

The bottom of Figure 1 shows a simplified diagram of the task. Blocks represent the camera operator's control behavior, and the camera and tripod's rotational dynamics. The athlete's direction relative to the tripod is the reference signal, $r(t)$, the camera's actual direction is the camera state, $y(t)$, and the angle between the actual and desired directions is the error, $e(t)$. The operator's force

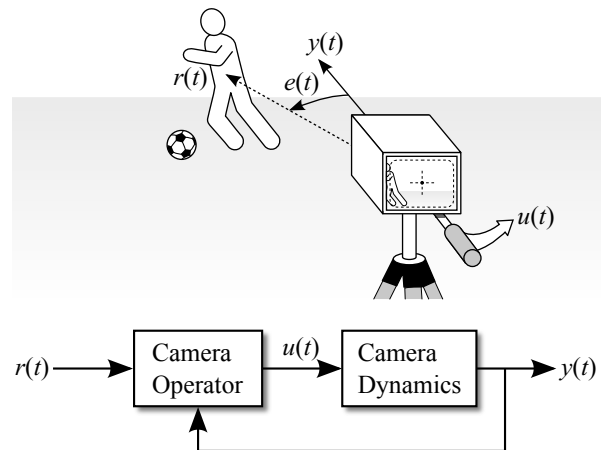


Figure 1. One-dimensional video camera tracking task.

on the handle is the command input, $u(t)$.

Note that to simulate this system, a model of the human's control behavior must be specified. Such models can be found in the field of *manual control*, which uses the tools and techniques of control theory to study the control behavior of humans. A Modelica library that captures knowledge from this field would be useful to modelers of human-machine systems.

This paper presents a library with models of human control behavior from the manual control literature. In addition, tools allow users to perform manual tracking tasks designed in Modelica, and to tune parameter values in the manual controller models to either maximize tracking performance, or to match recorded control input from user experiments. The next section gives background information about manual control and manual tracking tasks. Then, Sections 3 and 4 present the ManualTracking Modelica library and the supporting functions, respectively. Example tracking tasks are described in Section 5, and conclusions are drawn in Section 6.

2 Manual Tracking

Previous studies have made extensive use of single-axis manual tracking tasks to investigate the control behavior of humans performing continuous control. In a typical experimental tracking task, a human operator views

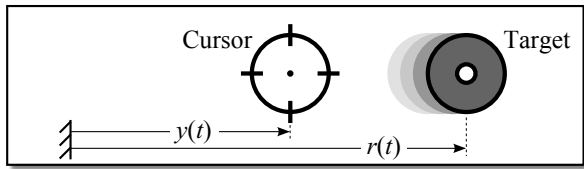


Figure 2. Display for manual tracking task.

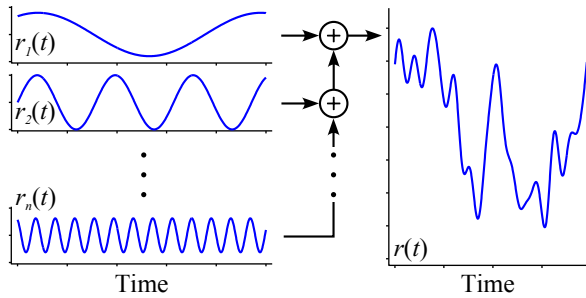


Figure 3. Sum of sines forcing function.

a display on a computer screen and uses an input device, such as a joystick or force stick, to generate control input. An example display is shown in Figure 2. There are two objects on the screen: one is a *target* that represents the reference (desired) state, and the other is a *cursor* that represents the actual state of the controlled system. The human’s goal is to make the cursor follow the target as closely as possible.

Many situations require humans to perform multi-axis, multi-loop control tasks, so it might seem that studying one-dimensional control would be an unreasonable oversimplification. However, it has been found that multi-axis tracking performance is highly related to one-axis tracking (Todosiev et al., 1967), and that information about the human controller derived from single-axis tracking tasks can be applied to multi-loop tasks (McRuer et al., 1975).

2.1 Forcing Functions

In the tracking display of Figure 2, the target’s motion is prescribed by a *forcing function*. This function should appear random to prevent the operator from predicting future behavior of the target, unless the real-world control task consists of highly predictable signals. This library, and much of manual control theory, focuses on the tracking of unpredictable signals.

From past studies, it has been shown that the sum of 5 or more sine waves is unpredictable to human operators (McRuer et al., 1965). An example summed-sine forcing function is shown in Figure 3. The individual sine waves on the left of Figure 3 are combined to yield the more complicated function on the right. In equation form,

$$r(t) = \sum_{i=1}^n A_i \sin(\omega_i t + \phi_i), \quad (1)$$

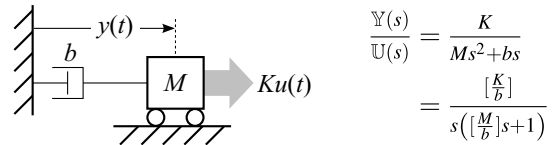


Figure 4. Mechanical example of a controlled element.

where A_i , ω_i , and ϕ_i are the sine wave amplitudes, frequencies, and phase angles, respectively. In general, low-frequency sine waves are given large amplitudes, and waves with increasing frequency are given increasingly small amplitudes (Jagacinski and Flach, 2003). The difficulty of tracking a given forcing function depends heavily on the velocity and acceleration of the target motion (Damveld et al., 2010).

2.2 Controlled Elements

The *controlled element* is the dynamic response of the cursor to control input, and it represents the real-world system under human control. A simple mechanical example is shown in the left side of Figure 4. A rolling cart with mass M is attached to ground by a damper with damping coefficient b , and the control input pushes the cart with a force of magnitude $Ku(t)$. The equivalent controlled-element transfer function is shown in the right side of Figure 4. The cart exhibits a lagged velocity response with time constant M/b and steady-state velocity K/b . The units of these parameters depend on the units chosen for M , b , and K .

Simple models have been used to capture the primary behavior of certain degrees of freedom in aircraft (McRuer and Jex, 1967), automobiles, and other complicated systems. Many experiments have used the simplest controlled elements with position, velocity, and acceleration responses.

2.3 Manual Controllers

Human control behavior while tracking an unpredictable signal can be modeled using tools and techniques from control theory. A specific model will be called a *manual controller* model. These models are generally either structural or algorithmic in nature (McRuer, 1980). Structural models use explicit equations and parameters to model human control pathways and the human’s resulting input-output response. Algorithmic models use a more implicit optimal control formulation, where only the human’s total response is computed. This library includes only structural models. For a review of both kinds of models, see Hess (2006).

Structural manual controller models have taken many forms, but most include one or more of the control pathways shown in Figure 5. Nearly all controllers include the *compensatory* pathway, which acts on the error $e(t)$ between the reference and measured state. Manual track-

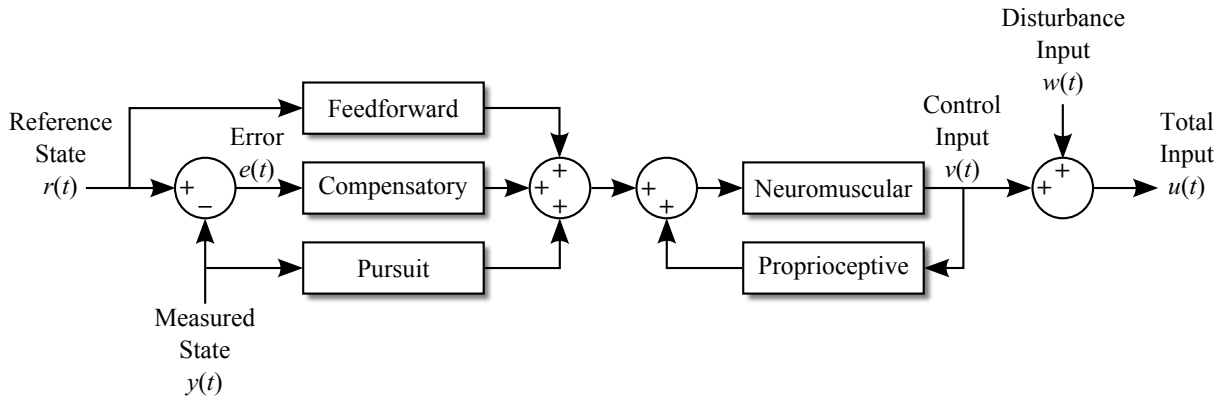


Figure 5. Manual controller signals and control pathways.

ing experiments that display only this error, and not the reference and measured states independently, are called compensatory tracking tasks.

If both the reference state and the measured state are displayed to the human, then they can be used for the *feedforward* and *pursuit* control actions. The presence of pursuit information does not guarantee pursuit control will be used, and the absence of pursuit information does not guarantee pursuit control will not be used.

The *neuromuscular* filter accounts for the lag imposed by limb dynamics and neuromuscular delays. The human senses the filtered input using the *proprioceptive* pathway, and compares it to the desired input.

Once the human's control input is determined, a *disturbance input* is added. This can be used for a disturbance rejection task (Van Paassen and Mulder, 2006), or to add *remnant* to the controller model. Remnant accounts for the human's control input that is not predicted by the model. Most of the remnant appears to come from fluctuations in the effective time delay (McRuer, 1980), nonsteady control behavior, and nonlinear anticipation or relay-like operations (McRuer et al., 1967). These effects are larger when tracking conditions are difficult (Hess, 1979). The remnant has been found to have fairly constant power with no major peaks, and it tends to be relatively small when tracking conditions are favorable (Wade and Jex, 1972).

Perhaps the most influential model has been the *Crossover model* proposed by McRuer and Jex (1967). The Crossover model states that in a compensatory task (when only $e(t)$ is displayed) for a variety of controlled-element dynamics, the operator acts to make the overall human-machine system assume the form:

$$\frac{\mathbb{Y}(j\omega)}{\mathbb{E}(j\omega)} = \frac{Ke^{-j\omega\tau}}{j\omega} \quad \text{near } \omega = K, \quad (2)$$

where K is the open-loop system gain, and τ is the effective time delay. Note that the transfer function is written with the frequency operator $j\omega$ instead of the Laplace variable s . This is to emphasize that the model is only intended to apply in the frequency domain, and may not be

accurate for non-sinusoidal inputs such as steps or ramps. Furthermore, the Crossover model is only meant to characterize the system near the crossover frequency – hence its name. The control system's closed-loop response is generally dominated by its behavior near the crossover frequency (McRuer and Jex, 1967).

Note that control input does not appear explicitly in the Crossover model. It is an implicit model that depends on the controlled-element dynamics. Therefore it is not included in the ManualTracking library, which requires explicit models to generate the control input. However, many of these explicit models were originally formed using the Crossover model as a basis.

3 Modelica Library

The previous section introduced the elements of typical manual tracking tasks, and this section presents their implementation in the ManualTracking Modelica library. An overview of the library structure is shown in Figure 6. Packages in the library will be described in order from least to most complex, ending with the TrackingTasks package. An instance of a tracking task requires instances from the ManualControllers, ControlledElements, and ForcingFunctions packages. The Blocks and Icons packages are straightforward, and will not be discussed.

ForcingFunctions

This package only includes the summed sine wave signal, which is by far the most common signal used in manual tracking tasks. Frequency values can be either in units of Hz (with SumOfSinesHz), or radians per second (with SumOfSinesRadPerSec). If the user wishes to make a custom forcing function for either the reference signal or disturbance input, the signal must be contained in one block with a single output, and it must be stored in the appropriate package. If these rules are violated, the Python functions will not be able to parse the text of the tracking task.

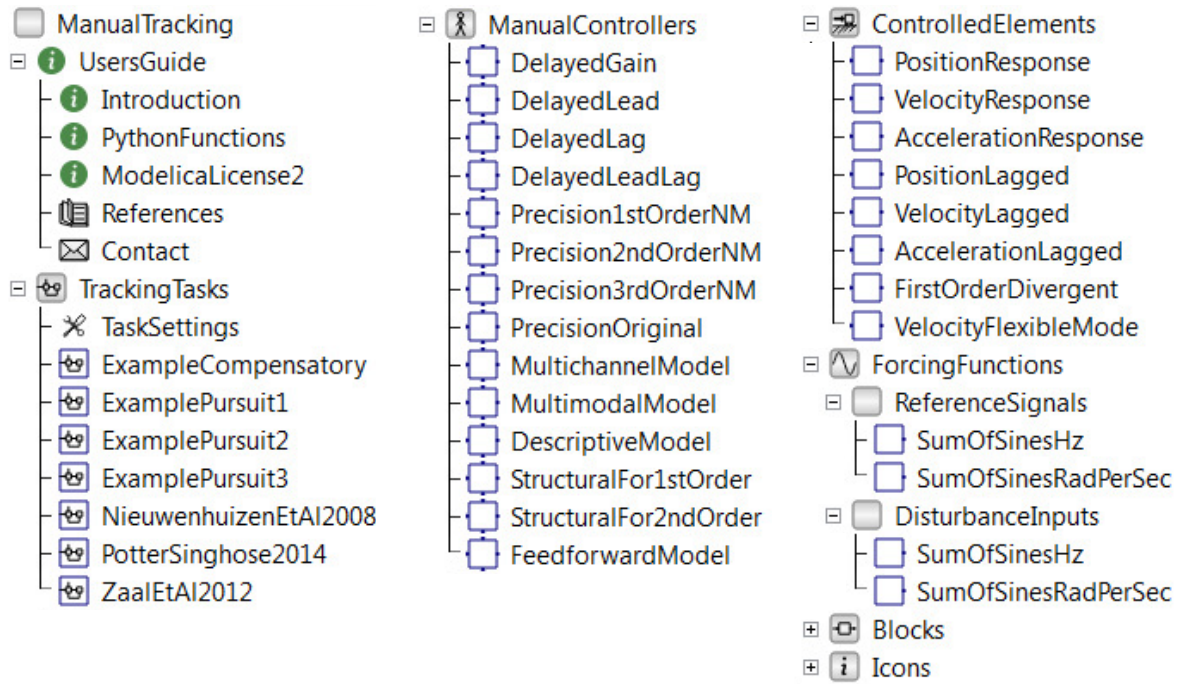


Figure 6. ManualTracking library overview.

ControlledElements

All controlled elements included in the library are shown in Table 1. There are the basic position, velocity, and acceleration responses. There are also versions of these basic responses with an added first-order lag, making them less responsive at first, but eventually reaching the same steady-state position/velocity/acceleration.

The unstable FirstOrderDivergent controlled element was used to investigate limitations of a human’s effective time delay in Jex et al. (1966). The VelocityFlexibleMode includes a second order mode that can be oscillatory. This controlled element was studied with relatively high damping in Shirley and Young (1968), and very low damping in Potter and Singhose (2014).

ManualControllers

Table 2 shows all manual controller models in equation form. The ManualTracking library documentation contains block diagrams of each controller model, and these block diagrams are sometimes more intuitively useful than the equations.

McRuer and Jex (1967) describe how the first four manual controllers are combined with specific controlled elements to yield the form of the Crossover model. The model PrecisionOriginal was proposed by McRuer and his colleagues, and various simplified versions with one of the lead-lag terms removed have been used since then. These versions mainly differ in how they represent the human’s neuromuscular filter. The MultichannelModel, MultimodalModel, and DescriptiveModel each include

Table 1. Controlled elements.

Class Name	Transfer Function
PositionResponse	K
VelocityResponse	K/s
AccelerationResponse	K/s^2
PositionLagged	$K/(Ts + 1)$
VelocityLagged	$K/(s[Ts + 1])$
AccelerationLagged	$K/(s^2[Ts + 1])$
FirstOrderDivergent	$K/(Ts - 1)$
VelocityFlexibleMode	$K\omega^2/(s[s^2 + 2\zeta\omega s + \omega^2])$

pursuit control, but they have different ways of organizing the human’s control pathways.

The following two models, StructuralFor1stOrder and StructuralFor2ndOrder, include proprioceptive feedback, and are designed to control first-order and second-order controlled elements, respectively. The FeedforwardModel includes feedforward control. For this control pathway, an inverted model of the plant dynamics is needed, and the controller automatically uses the block ManualTracking.Blocks.FeedForward for this purpose. To change the inverse dynamics block, or to make the controller use a different block, the Modelica file text must be modified manually.

Once the controller is selected, the next task is to choose values for controller parameters. Tools described in Section 4 can help select these values – they are selected either to yield optimal performance, or the closest fit to experimental control behavior.

Table 2. Manual controller models.

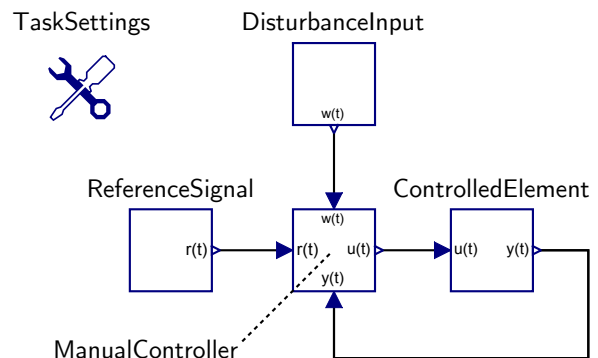
Class Name	Control Input, $\mathbb{V}(s)$ as a function of Laplace-transformed $r(t)$, $y(t)$, and $e(t)$, defined in Figure 5
DelayedGain	$\mathbb{E}(K)e^{-\tau s}$
DelayedLead	$\mathbb{E}(K(Ts+1))e^{-\tau s}$
DelayedLag	$\mathbb{E}\left(K\frac{1}{T_2s+1}\right)e^{-\tau s}$
DelayedLeadLag	$\mathbb{E}\left(K\frac{T_1s+1}{T_2s+1}\right)e^{-\tau s}$
Precision1stOrderNM	$\mathbb{E}\left(K\frac{T_1s+1}{T_2s+1}\right)\left(\frac{1}{T_3s+1}\right)e^{-\tau s}$
Precision2ndOrderNM	$\mathbb{E}\left(K\frac{T_1s+1}{T_2s+1}\right)\left(\frac{\omega^2}{s^2+2\zeta\omega s+\omega^2}\right)e^{-\tau s}$
Precision3rdOrderNM	$\mathbb{E}\left(K\frac{T_1s+1}{T_2s+1}\right)\left(\frac{\omega^2}{(T_3s+1)(s^2+2\zeta\omega s+\omega^2)}\right)e^{-\tau s}$
PrecisionOriginal (McRuer et al., 1965)	$\mathbb{E}\left(K\frac{T_1s+1}{T_2s+1}\right)\left(\frac{T_3s+1}{T_4s+1}\right)\left(\frac{\omega^2}{(T_5s+1)(s^2+2\zeta\omega s+\omega^2)}\right)e^{-\tau s}$
MultichannelModel (Nieuwenhuizen et al., 2008)	$\left[\mathbb{E}(K_1(T_1s+1))e^{-\tau_1s} + \mathbb{Y}\left(K_2\frac{s^2(T_2s+1)}{T_3s+1}\right)e^{-\tau_2s}\right]\frac{\omega^2}{s^2+2\zeta\omega s+\omega^2}$
MultimodalModel (Zaal et al., 2012)	$\left[\mathbb{E}\left(K_1\frac{(T_1s+1)^2}{T_2s+1}\right)e^{-\tau_1s} + \mathbb{Y}(K_2s)e^{-\tau_2s}\right]\frac{\omega^2}{s^2+2\zeta\omega s+\omega^2}$
DescriptiveModel (Hosman and Stassen, 1999)	$\left[\mathbb{E}(K_1e^{-\tau_1s} + K_2se^{-\tau_2s}) + \mathbb{Y}\left(K_3se^{-\tau_3s} + K_4\frac{s^2(T_1s+1)}{(T_2s+1)(T_3s+1)}\right)\right]e^{-\tau_4s}$
StructuralFor1stOrder (Hess, 1980)	$\mathbb{E}(K_1 + K_2se^{-\tau_1s})\left(\frac{\omega^2(Ts+1)}{\omega^2K_3s+(s^2+2\zeta\omega s+\omega^2)(Ts+1)}\right)e^{-\tau_2s}$
StructuralFor2ndOrder (Hess, 1980)	$\mathbb{E}(K_1 + K_2se^{-\tau_1s})\left(\frac{\omega^2(T_1s+1)(T_2s+1)}{\omega^2K_3s+(s^2+2\zeta\omega s+\omega^2)(Ts+1)(T_2s+1)}\right)e^{-\tau_2s}$
FeedforwardModel (Drop et al., 2013)	$\left[\mathbb{E}(K_1)e^{-\tau_1s} + \mathbb{R}\left(K_2\frac{1}{T_3s+1}\right)[\text{FeedForward}]e^{-\tau_2s}\right]\frac{\omega^2}{s^2+2\zeta\omega s+\omega^2}$

TrackingTasks

Blocks from the ForcingFunctions, ControlledElements, and ManualControllers packages can be useful by themselves, in any configuration that supports the user's model. However, to use the parameter tuning and user experiment features of this library, a specific configuration of the components is required.

A tracking task model should have the standard form shown in Figure 7, and it should be stored inside the TrackingTasks package. It includes one block from the ForcingFunctions.ReferenceSignals, ForcingFunctions.DisturbanceInputs, and ControlledElements packages, each connected to the appropriate port of a ManualController block. Several manual tracking tasks from the literature are provided.

An additional component, the TaskSettings block, must be included. This block contains important details of the tracking task: **taskDuration** is the total length of the task; **previewTime** is the amount of time in advance


Figure 7. Example of tracking task.

to show the target motion; and **backgroundVisible** determines whether or not pursuit (background) information is shown with hatch marks. The last three parameters are only used in the user experiment, and not in the parameter tuning or simulation functions.

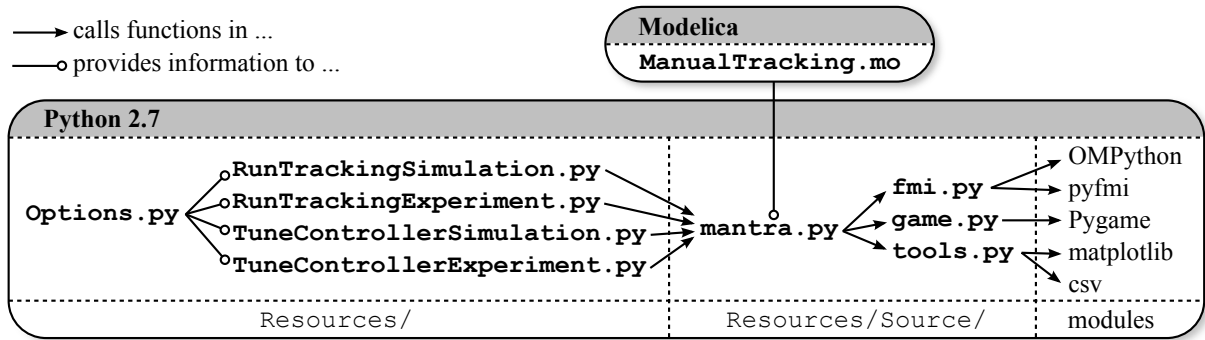


Figure 8. Software overview.

4 Python Functions

The previous sections have described purely Modelica-based components that can be run from within a Modelica simulation environment. Two additional capabilities are provided in the `ManualTracking` library: automatically tuning manual controller parameters, and performing real-time tracking experiments. These features are implemented in the Python programming language.

An overview of the software is shown in Figure 8. In the `Resources/` directory, there are 5 `.py` files. Four of these are function scripts that can be run from a terminal or Python IDE, and the fifth file is `Options.py`, where options are set by the user. The functions do not use input arguments, and instead get them from `Options.py`. Variables in this file include: `taskModel`, the tracking task model to run; `saveFormat`, the format with which to save backup data files; `printVerbose`, whether or not to print all runtime messages to the console. The user may also experiment with different framerates and optimization/simulation methods.

`Options.py` also contains Boolean input arguments for each of the four functions: `useSaved` makes the functions use most recent saved data (in `Resources/Temp/` directory) instead of re-running an experiment; `plotResults` generates a figure with the resulting trajectories; and `saveResults` saves a backup results file in the `Resources/Data/` directory. The results file contains values for the reference state, measured state, disturbance input, and control input at each sample time.

Each of the four main functions call `mantra.py`, which reads the `ManualTracking.mo` file for details of the tracking task, and then calls functions defined in `fmi.py`, `game.py`, and `tools.py`. These functions use standard Python modules, as well as `OMPpython`, `pyfmi`, `Pygame`, and `matplotlib`. Note that `OMPpython` requires an installation of OpenModelica (Fritzson et al., 2005). Additionally, `pyfmi` has many dependencies that must be installed first, and it may be more convenient to install `JModelica.org` (Åkesson et al., 2010) instead of installing them individually. After all required Python modules have been installed, the following functions should run successfully.

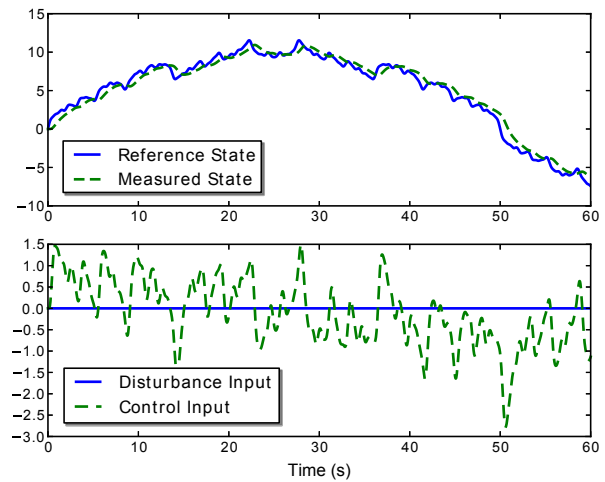


Figure 9. Plot of simulation results.

4.1 RunTrackingSimulation

This function simulates the tracking task model specified in `Options.py`. The simulation stop time is specified by `taskDuration` in the `TaskSettings` block. All files are saved into the `Resources/Temp/` directory. Generated files include log files, FMU build files, and a comma-separated value (CSV) file of simulation results. If `saveResults` is true, this CSV file is also saved in the `Resources/Data/` directory. If `plotResults` is true, then time-curves of $r(t)$, $y(t)$, $w(t)$, and $u(t)$ are plotted, as shown in Figure 9. The Python module `matplotlib` is required for this feature.

4.2 RunTrackingExperiment

This function allows the user to perform a tracking task in real time. The tracking task model specified in `Options.py` is parsed, and details of the experiment are extracted from the `TaskSettings` block. Next, the reference signal and disturbance input are generated by building them into separate FMUs, and simulating them for the duration of the experiment. These signals are not affected by the user input or controlled element state, and therefore they can be simulated open loop.

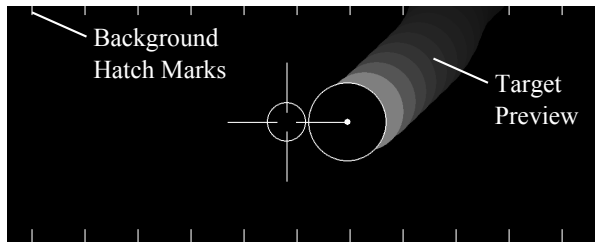


Figure 10. Display of manual tracking experiment.

Next, Pygame looks for any joysticks connected to the computer. If no joystick is found, then the keyboard arrow keys may be used for control input. When a joystick is used, the experiment runs more smoothly and the parameter-fitting functions work much more effectively. Therefore, using a joystick for the experiment is highly recommended.

Then, two scaling factors are automatically calculated. One is the *display gain*, which determines how far to move display objects (in pixels) based on the reference and measured state magnitudes, which are in unknown units of length. The other is the *input gain*, which determines the magnitude of input to the controlled element based on the joystick or keyboard input between -1 (full left) and 1 (full right).

Finally, the user is prompted to start the experiment. The tracking display is shown in Figure 10. Lines on the top and bottom of the screen mark the global coordinates, so that the target motion can be seen independently of the cursor motion. These lines can be hidden to create a compensatory task by setting `backgroundVisible` to false in the `TaskSettings` block.

Figure 10 also shows a preview of the target motion. Future motion is indicated by circles falling from the top of the screen. The topmost circle shows where the target will be `previewTime` seconds in the future. This feature can be disabled by setting `PreviewTime` to 0.

If desired, the user may adjust fundamental settings of the game in the `Resources/Source/game.py` file. Modifying these settings may cause errors, so it is a good idea to save a backup of the `game.py` file before making modifications.

4.3 TuneControllerSimulation

This function repeatedly simulates the tracking task with different parameter values in the manual controller, and finds values which yield the best tracking performance. This reflects an important finding in the literature: an experienced human operator has inherent human limitations¹ but behaves in a nearly optimal fashion given these limitations.

Mathematically, the function tries to minimize the integrated squared difference between $y(t)$ and $r(t)$. This

¹For example, reaction time delay, neuromuscular lag, and ability to generate derivatives and higher-order leads.

is shown conceptually in Figure 11(a), where $c(t)$ is the continuous cost to minimize. Because tracking performance is not a differentiable function of the controller parameters, a derivative-free optimization method such as the Nelder-Mead simplex method (Gedda et al., 2012) must be used.

Some manual controller models contain many parameters, and attempting to tune all of them at once would be time-consuming and would likely yield poor results. Therefore, the user is allowed to select a subset of the parameters using a console prompt like this:

```
Tunable controller parameters:
1. K -- Proportional gain
2. T2 -- Time constant of phase
   lag compensation (s)
3. T1 -- Time constant of phase
   lead compensation (s)

Please enter a comma-separated number
list specifying parameters to tune: _
```

To tune `K` and `T1`, for example, the user should type `1,3` and press `Enter`. The rest of the parameters are fixed so that they remain the same as in the manual controller component definition, unless they are re-assigned in the tracking task model.

4.4 TuneControllerExperiment

This function tunes the automatic manual controller to behave as much as possible like the human controller. The concept is shown in Figure 11(b). The goal is to minimize the difference between the experimentally recorded control input, $u(t)$, and the simulated control input, $\hat{u}(t)$.

The input to the manual controller is the reference signal (and disturbance input, not shown in Figure 11), and the experimentally recorded controlled element state. Note that the tracking performance of the tuned controller might be very poor, because it does not attempt to optimize tracking performance. It simply tries to match control behavior of the user.

4.5 Notes

- When selecting a controller model, one should consider details of the real-world task and controlled-element dynamics. The Python functions do not check the appropriateness of a manual controller for the given tracking conditions.
- User-created `ReferenceSignal`, `DisturbanceInput`, `ControlledElement`, and `ManualController` classes must be stored inside the appropriate packages. The Python functions only search in these locations when parsing the tracking task.

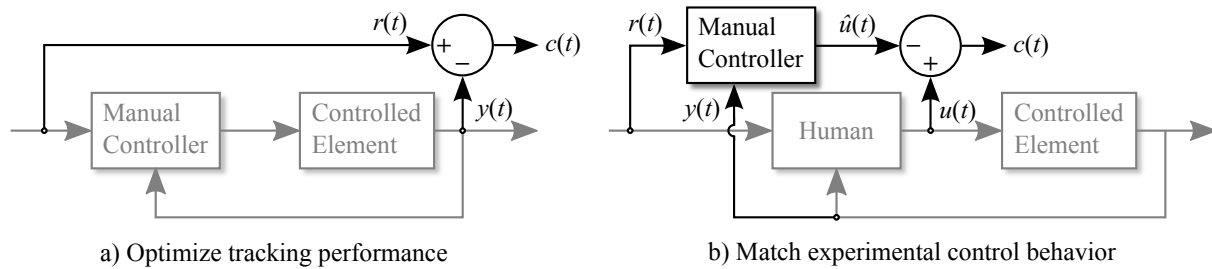


Figure 11. Tuning manual controller parameters by minimizing $\int [c(t)]^2 dt$.

- While the keyboard can be used for control input in the tracking experiment, a joystick is highly recommended. The parameter tuning functions work much better, and the display is smoother.
- Automatic syncing programs (Dropbox in particular, but possibly others) seem to cause a problem with the real-time user experiment. Try exiting, or at least pausing, these programs if the experiment crashes repeatedly.

5 Example

Basic use of the Modelica library and Python functions will now be demonstrated. Load the ManualTracking library, go to the TrackingTasks package, and simulate one of the example tasks. Plots of `controlledelement1.y` and `referencesignal1.y` should resemble the top plot of Figure 9, with the controlled element following the reference signal closely, but with a small time delay.

Next, navigate to `ManualTracking/Resources/`, and open `Options.py`. The `taskModel` variable should be assigned to one of the example tracking tasks, and `printResults` should be set to true. Then run the script `RunTrackingSimulation.py`, either from a Python IDE or from a console window. A few diagnostic messages should print to the console, and then a figure similar to Figure 9 should appear. To reduce the amount of console output, set `printVerbose` to false.

If the function executed successfully, then try the real-time experiment. Run `RunTrackingExperiment.py` and wait for the reference signal and disturbance inputs to be generated. After a short time, the console should show this prompt:

```
Press 'Enter' to bring up the display,
then press any key except 'q' to start
the experiment: _
```

After following these instructions, a window similar to Figure 10 appears. Use either the arrow keys or a joystick to make the crosshairs follow the target. Once the experiment is complete, a plot of the state and input trajectories is shown if `printResults` is set to true. To examine the data file used

for this plot, go to the `Resources/Temp/` directory, and look for the comma-separated-value (CSV) file `ExampleTaskName_exp.csv`. The data file for the tracking simulation should also be in the same directory, saved as `ExampleTaskName_sim.csv`.

Next, try the manual controller tuning functions. Run the `TuneControllerSimulation.py` script. When prompted, type 1, 3 and press enter. The optimization function prints information about the current parameter guesses and cost function value to the console. Within a few minutes, the solver should converge, and parameter values for the parameters should be displayed. A plot shows the simulated tracking performance using these parameter values.

Instead of tuning parameters to yield the best tracking performance, they can be tuned to fit experimental tracking performance. First, make sure `useSaved` is set to true in `Options.py`, otherwise the user experiment will be run again. Then run the `TuneControllerExperiment.py` script. Just like in the previous example, select the parameters you would like to tune, and the function should find their optimal values and display the simulated tracking results with the chosen controller values.

6 Conclusions

This paper presents a Modelica library and supporting functions for studying human control behavior in continuous tracking tasks. These tools can increase Modelica's usefulness for modeling human-machine systems. For now, further development, debugging, and testing on different platforms is needed. The library is open source and available for download at <http://jjpotterkowski.github.io/>.

Acknowledgements

This work was fully supported by an ERCIM "Alain Bensoussan" post-doctoral research fellowship, hosted by VTT Technical Research Centre of Finland. The author wishes to thank A. Ashgar, A. Pop, and M. Sjölund for technical advice related to OMPython and FMUs.

References

- G. A. Bekey, G. O. Burnham, and J. Seo. Control theoretic models of human drivers in car following. *Human Factors*, 19(4):399–413, Aug. 1977.

- H. J. Damveld, G. C. Beerens, M. M. van Paassen, and M. Mulder. Design of forcing functions for the identification of human control behavior. *AIAA Journal of Guidance, Control, and Dynamics*, 33(4):1064–1081, Jul.-Aug. 2010.
- F. M. Drop, D. M. Pool, H. J. Damveld, M. M. van Paassen, and M. Mulder. Identification of the feedforward component in manual control with predictable target signals. *IEEE Transactions on Cybernetics*, 43(6):1936–1949, Dec. 2013.
- P. Fritzson, P. Aronsson, H. Lundvall, K. Nyström, A. Pop, L. Saldamli, and D. Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. *Simulation News Europe*, 44(45), Dec. 2005.
- S. Gedda, C. Andersson, J. Åkesson, and S. Diehl. Derivative-free parameter optimization of functional mock-up units. In *Proc. 9th Int. Modelica Conf.*, Munich, Germany, Sep. 2012.
- R. A. Hess. A rationale for human operator pulsive control behavior. *Journal of Guidance and Control*, 2(3):221–227, May-Jun. 1979.
- R. A. Hess. A structural model of the adaptive human pilot. *AIAA Journal of Guidance, Control, and Dynamics*, 3(5):416–423, Sep.-Oct. 1980.
- R. A. Hess. *Feedback Control Models – Manual Control and Tracking*, chapter 38, pages 1249–1294. John Wiley & Sons, Inc., Hoboken, NJ, 3 edition, 2006.
- R. A. Hess and A. Modjtahedzadeh. A control theoretic model of driver steering behavior. *IEEE Control Systems Magazine*, 10(5):3–8, Aug. 1990. doi:10.1109/37.60415.
- R. Hosman and H. Stassen. Pilot's perception in the control of aircraft motions. *Control Engineering Practice*, 7:1421–1428, 1999.
- R. J. Jagacinski. A qualitative look at feedback control theory as a style of describing behavior. *Human Factors*, 19:331–347, Aug. 1977.
- R. J. Jagacinski and J. M. Flach. *Control Theory for Humans: Quantitative Approaches to Modeling Performance*. CRC Press, New York, NY, 2003.
- H. R. Jex, J. D. McDonnell, and A. V. Phatak. A “critical” tracking task for manual control research. *IEEE Transactions on Human Factors in Electronics*, HFE-7(4):138–145, Dec. 1966. doi:10.1109/THFE.1966.232660.
- 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*, 34(11):1737–1749, Nov. 2010.
- D. L. Kleinman and T. R. Perkins. Modeling human performance in a time-varying anti-aircraft tracking loop. *IEEE Transactions on Automatic Control*, AC-19(4):297–306, Aug. 1974.
- D. T. McRuer. Human dynamics in man-machine systems. *Automatica*, 16(3):237–253, May 1980.
- D. T. McRuer and H. R. Jex. A review of quasi-linear pilot models. *IEEE Transactions on Human Factors in Electronics*, HFE-8(3):231–249, Sep. 1967. doi:10.1109/THFE.1967.234304.
- D. T. McRuer, D. Graham, E. S. Krendel, and W. Reisner. Human pilot dynamics in compensatory systems. Technical Report AFFDL-TR-65-15, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, OH, 1965.
- D. T. McRuer, D. Graham, and E. S. Krendel. Manual control of single-loop systems: Part I. *Journal of the Franklin Institute*, 283(1):1–29, Jan. 1967.
- D. T. McRuer, D. H. Weir, H. R. Jex, R. E. Magdaleno, and R. W. Allen. Measurement of driver-vehicle multiloop response properties with a single disturbance input. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-5(5):490–497, 1975. doi:10.1109/TSMC.1975.5408371.
- F. M. Nieuwenhuizen, P. M. T. Zaal, M. Mulder, M. M. van Paassen, and J. A. Mulder. Modeling human multichannel perception and control using linear time-invariant models. *Journal of Guidance, Control, and Dynamics*, 31(4):999–1013, Jul.-Aug. 2008.
- J. J. Potter and W. E. Singhose. Effects of input shaping on manual control of flexible and time-delayed systems. *Human Factors*, 56(7):1284–1295, Nov. 2014.
- R.S. Shirley and L.R. Young. Motion cues in man-vehicle control: effects of roll-motion cues on human operator's behavior in compensatory systems with disturbance inputs. *IEEE Transactions on Man-Machine Systems*, 9(4):121–128, Dec. 1968.
- E. P. Todosiev, R. E. Rose, and L. G. Summers. Human performance in single and two-axis tracking systems. *IEEE Transactions on Human Factors in Electronics*, HFE-8(2):125–129, Jun. 1967.
- A. Tustin. The nature of the operator's response in manual control, and its implications for controller design. *Journal of the Institution of Electrical Engineers*, 94(2):190–206, May 1947.
- M. M. Van Paassen and M. Mulder. *International Encyclopedia of Ergonomics and Human Factors*, volume 1, chapter Identification of Human Control Behavior, pages 400–407. Taylor and Francis, London, 2 edition, 2006.
- A. R. Wade and H. R. Jex. A simple Fourier analysis technique for measuring the dynamic response of manual control systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2(5):638–643, Nov. 1972. doi:10.1109/TSMC.1972.4309192.
- P. M. T. Zaal, D. M. Pool, M. M. van Paassen, and M. Mulder. Comparing multimodal pilot pitch control behavior between simulated and real flight. *Journal of Guidance, Control, and Dynamics*, 35(5):1456–1471, Sep.-Oct. 2012.

Model-based control with FMI and a C++ runtime for Modelica

Rüdiger Franke¹ Marcus Walther² Niklas Worschech³ Willi Braun⁴ Bernhard Bachmann⁴

¹ABB, ruediger.franke@de.abb.com, ²TU Dresden, marcus.walther@tu-dresden.de,

³Bosch Rexroth, niklas.worschech@boschrexroth.de,

⁴FH Bielefeld, {willi.braun, bernhard.bachmann}@fh-bielefeld.de

Abstract

Modelica describes physical systems on a high level, using model objects, multi-dimensional arrays and other data structures as well as graphical representations. Modelica models are translated to differential-algebraic equation systems and compiled to executable code prior to their execution in numerical solvers. The translation gives a lot of possibilities for code optimization. This is particularly important for model-based control applications.

This paper investigates the exploitation of C++ for Modelica code optimization. C++ supports advanced programming concepts and at the same time aims to “leave no room for a lower-level language ... (except for assembly code in rare cases)” (B. Stroustrup: The C++ Programming Language, 2014). The features exploited here include polymorphism, templates, built-in exception handling and object destructors.

The ideas have been implemented in the OpenModelica C++ runtime. The paper describes its enhancement with new array features and with an FMI 2.0 interface. FMI serves as interface between modeling tools and control applications. In particular the new FMI 2.0 meets requirements of numerical optimization solvers in model-based control.

A publicly available application example demonstrates the achievements. CPU times obtained with the OpenModelica C++ runtime are significantly faster than CPU times obtained with the C runtime or with Dymola.

Keywords: Modelica, OpenModelica, FMI, C++, model-based control, MPC, MHE, SQP, HQP.

1 Introduction

The development of the Functional Mock-up Interface (FMI) was originally driven by automotive industries. The goal was to improve simulation model exchange between component suppliers and OEMs during product development. The FMI standard supports model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code (FMI, 2014).

FMI 2.0 for model exchange introduces major enhancements, like sparse model structures and directional derivatives. These enhancements make FMI applicable beyond functional mock-ups for model-based control and optimization as well, evolving it to a Functional Model Interface.

Real-time control applications pose further requirements, like small code size, high quality of generated binaries and fast execution speed. The C++ runtime of OpenModelica is focusing on real-time requirements (Worschech and Mikelsons, 2012). This makes it superior for model-based control applications.

It must be noted that the real-time applications addressed here require cycle times of seconds, sometimes going down to milliseconds or up to minutes. We assume an execution platform with relatively high performance, starting from devices like Raspberry PI and ranging up to distributed server farms. We don't consider smaller devices with only few kilobytes of memory or lacking floating point arithmetic, because we rate engineering efficiency exploiting high-level technologies like Modelica or C++ more important than extreme hardware savings.

2 Model-based control with mathematical programs

Modelica models are typically used for initial-value simulations. The strict separation of Modelica models from numerical solvers opens further application areas. The models may serve as constraints in mathematical programs as well. Mathematical programming is a technology to solve tasks described with constraints and objective function. This section outlines how Modelica and Mathematical Programming are brought together for model-based control.

2.1 Related work and design rationale

The Optimization Library developed by DLR adds a numerical optimization solver to the Dymola modeling and simulation environment (Pfeiffer, 2012). The focus is on usability, supporting engineering design simulations. A mathematical program is formulated with optimization

attributes, like bounds or weights, which are added to a readily compiled simulation model using regular Modelica parameter dialogs. The idea of custom attributes has been generalized as custom annotations (Zimmer et al, 2014).

Alternatively to adding an optimization specification on top of a simulation model, there has been an attempt to extend the Modelica syntax with the Optimica language (Åkesson et al, 2010). The approach is conceptionally questionable because it mixes the physical modeling language Modelica with mathematical programs. It is claimed that this improves the treatment of large-scale optimization programs for optimal control. The optimization specific extensions of the Modelica language hinder the re-use of simulation technologies like FMI for optimization. Work basing on Optimica typically involves the development of specific complex tool chains. Results published so far show feasibility, but no advantages over earlier simpler approaches, see e.g. (Lazutkin et al, 2014; Magnusson et al, 2014; Ruge et al, 2014). Recent publications report a convergence towards simpler approaches that re-use simulation technologies, like BLT transformation (Magnusson et al, 2014; Ruge et al, 2014).

The optimization approach used here was developed and first published many years ago. It combines the advantages of optimization formulations using custom attributes with the efficient treatment of large-scale optimization programs for optimal control. A front-end for the optimization solver HQP (Franke and Arnold, 1997) converts optimal control problems formulated for simulation models to large-scale nonlinear programs treated internally. The design rationale was to re-use existing modeling and simulation technologies for optimization, minimizing additional development effort and dependencies on specific tools. The new FMI standard fits well into the long standing design rationale.

Meanwhile there find many industrial applications of HQP, including the control of water canal systems (Wagenpfeil et al, 2014), boom cranes (Neupert et al, 2010) and polymerization reactors (Nagy et al, 2007). HQP has been integrated with the ABB control system and is being applied to the model-based optimal control of power plants worldwide since a decade (Franke and Vogelbacher, 2006; Franke et al, 2008). Recent applications address the real-time optimization of large numbers of renewable power units in virtual power plants and smart grids (Franke et al, 2014).

2.2 Treating model-based control with mathematical programs

Many advanced model-based control applications can be treated as mixed discrete/continuous optimal control problems. Examples include moving horizon estimation (MHE) and model predictive control (MPC).

Discrete-time model equations result from the implementation of control systems on digital computers with cyclically running tasks. They are described with difference equations of the form

$$\begin{aligned} \mathbf{x}_d(\mathbf{k} + \mathbf{1}) &= \mathbf{f}_d[\mathbf{k}, \mathbf{x}_d(\mathbf{k}), \mathbf{x}_c(t_k), \mathbf{u}_d(\mathbf{k})], \\ \mathbf{x}_d(\mathbf{0}) &= \mathbf{x}_{d0}, \quad \mathbf{k} = \mathbf{0}, \mathbf{1}, \dots, \mathbf{K} \end{aligned} \quad (1)$$

Here $\mathbf{x}_d(\mathbf{k})$ are the discrete-time states at interval k with the corresponding sample time points t_k , $t_0 < t_1 < \dots < t_K$. The control inputs $\mathbf{u}_d(\mathbf{k})$ are optimized per sample time point. Optimized model parameters can be treated as additional states that are constant and have free initial values.

The continuous-time states $\mathbf{x}_c(t)$ describe physical processes, like devices for energy conversion or storage. They are defined with continuous-time differential equations of the form

$$\begin{aligned} \frac{d\mathbf{x}_c(t)}{dt} &= \mathbf{f}_c[t, \mathbf{x}_d(\mathbf{k}(t)), \mathbf{x}_c(t), \mathbf{u}_c(t)], \\ \mathbf{x}_c(t_0) &= \mathbf{x}_{c0}, \quad t \in [t_0, t_K] \end{aligned} \quad (2)$$

Numerical solvers generally require the parameterization of continuous-time trajectories $\mathbf{u}_c(t)$ with a finite number of control inputs $\mathbf{u}_c(\mathbf{k})$, such that $\mathbf{u}_c(t) = \mathbf{f}_u[t, \mathbf{u}_c(\mathbf{k}(t))]$. Typically the control inputs describe the control trajectories piecewise constant or piecewise linear.

The optimization has to consider physical and legal limitations that are formulated as constraints of the form

$$\mathbf{g}[t, \mathbf{x}_d(\mathbf{k}(t)), \mathbf{x}_c(t), \mathbf{u}_d(\mathbf{k}(t)), \mathbf{u}_c(t)] \geq \mathbf{0} \quad (3)$$

Remaining degrees of freedom are covered with the objective function

$$\sum_{k=0}^K f_0 \left[k, \begin{pmatrix} \mathbf{x}_d(k) \\ \mathbf{x}_c(t_k) \end{pmatrix}, \begin{pmatrix} \mathbf{u}_d(k) \\ \mathbf{u}_c(t_k) \end{pmatrix} \right] \rightarrow \min_{\substack{\mathbf{x}_d(0) \ \mathbf{u}_d(0) \\ \mathbf{x}_c(t_0)' \ \mathbf{u}_c(t_0)}} \quad (4)$$

Typical objectives are the minimization of costs or the maximization of results. Multiple objectives can often be expressed monetary and summed up.

2.3 The HQP solver

HQP treats mixed discrete/continuous optimal control problems as large-scale mathematical programs (Franke and Arnold, 1997). Continuous-time differential equations are approximated numerically over given discrete time intervals, either with fixed polynomials or using a variable step size solver. Discrete and continuous-time states and controls are combined into the state vector $x = (x_d, x_c)$ and the control vector $u = (u_d, u_c)$. This gives the discrete-time optimal control problem:

$$J = f_0(x^K) + \sum_{k=0}^{K-1} f_0(x^k, u^k) \rightarrow \min_{x^0, u^k}$$

with the discrete-time state equations

$$x^{k+1} = f^k(x^k, u^k), \quad k = 0, \dots, K-1$$

and the constraints

$$\begin{aligned} c^k(x^k, u^k) &\geq 0, \quad k = 0, \dots, K-1 \\ c^K(x^K) &\geq 0 \end{aligned} \quad (5)$$

The states and the control inputs of all time intervals are collected into one large vector of optimization variables

$$v = (x^0, u^0, x^1, u^1, \dots, x^{K-1}, u^{K-1}, x^K). \quad (6)$$

This results in the mathematical program

$$\begin{aligned} J(v) &\rightarrow \min_v & J: \mathbb{R}^n &\rightarrow \mathbb{R}^1 \\ h(v) &= 0 & h: \mathbb{R}^n &\rightarrow \mathbb{R}^{m_e} \\ g(v) &\geq 0 & g: \mathbb{R}^n &\rightarrow \mathbb{R}^m \end{aligned} \quad (7)$$

HQP treats large-scale nonlinear optimization with Sequential Quadratic Programming (SQP). Basing on the Lagrangian

$$\begin{aligned} L(v, \lambda, \mu) &= J(v) - \lambda^T h(v) - \mu^T g(v) \\ L: \mathbb{R}^n \times \mathbb{R}^{m_e} \times \mathbb{R}^m &\rightarrow \mathbb{R}^1 \end{aligned} \quad (8)$$

the solution must fulfill the Karush Kuhn Tucker (KKT) conditions

$$\begin{aligned} \nabla_v L(v, \lambda, \mu) &= \nabla J(v) - \nabla h(v)^T \lambda - \nabla g(v)^T \mu = 0 \\ \nabla_\lambda L(v, \lambda, \mu) &= -h(v) = 0 \end{aligned}$$

$$g(v) \geq 0$$

$$\mu \geq 0$$

$$g(v)^T \mu = 0$$

(9)

HQP applies Lagrange Newton iterations

$$\nabla^2 L(v, \lambda) \begin{pmatrix} \Delta v \\ \Delta \lambda \end{pmatrix} = -\nabla L(v, \lambda)$$

$$\begin{pmatrix} v^+ \\ \lambda^+ \end{pmatrix} := \begin{pmatrix} v + \Delta v \\ \lambda + \Delta \lambda \end{pmatrix}$$

(10)

to find the solution. The Lagrange Newton iteration is given here for the case $m=0$. HQP augments the Lagrangian to treat inequality constraints with an Interior Point method.

The Hessian of the Lagrangian $\nabla^2 L(v, \lambda, \mu)$ is formed numerically applying a rank 2 update in each time interval basing on the progress over subsequent iterations. This efficient multi-rank update is possible because the discrete-time model equations make the large-scale nonlinear program partial separable. There are only linear couplings between subsequent time intervals. This is also why no analytical second order derivatives are required.

The Jacobian of the Lagrangian $\nabla L(v, \lambda, \mu)$ is obtained by forming partial derivatives and by solving sensitivity equations along with the continuous-time differential equations of the model in each time interval. There exist different solvers, including fixed or variable step size and implicit or explicit.

2.4 Relation to other optimization approaches

The mathematical description given above basically applies to all optimization approaches mentioned in section 2.1. Only the following details differ:

1. The constraints (3) and optimization objective (4) may either be formulated as custom attributes for existing equations or as specific new equations for optimization (see Optimica).
2. The vector of optimization variables (7) contains optimized control variables and state variables. This results in large-scale sparse optimization programs. It has advantages if state constraints are present or for long time horizons. Alternatively the state variables may be hidden, resulting in smaller dense optimization programs (see the Optimization Library).
3. Explicit discrete-time state equations and constraints in (5) lead to partial separability, localizing non-linear terms inside individual time steps (also referred to as multiple shooting).

This gives the ability to apply efficient multi-rank updates to the numerical formation of information about second order derivatives. Alternatively the discrete-time equations may be replaced with nonlinear terms spanning two subsequent time steps (known as collocation and typically applied with Optimica). The lost partial separability may be compensated with analytic second order derivatives.

4. The nonlinear optimization program may be treated with a Newton method (typically used with Optimica basing on actual second order derivatives) or with a Quasi-Newton method basing on numerical updates (also known as SQP method and used with the Optimization Library and here).
5. Finally there finds different methods for the treatment of inequality constraints. Well known approaches include active set methods (see the Optimization Library) and interior point methods used with Optimica and here.

3 FMI for model based control

There exist a couple of different powerful Modelica tools, each having its particular pros and cons. FMI offers the advantage of being tool independent. This makes it possible to exploit the best features of different modeling tools depending on the application at hand. Once a control system can import FMI, it may be used together with any exporting tool.

Even with one and the same modeling tool the runtime code can be exchanged without effecting the optimization solver, e.g. from C code to C++ code as discussed below.

FMI 2.0 for model exchange covers many aspects of hooking a model to an optimization solver, like:

- Initialization mode for steady-state models
- Continuous-time mode for differential equations
- Change of parameter values at runtime
- Directional derivatives for Jacobian evaluation
- ModelStructure defining the sparse pattern in modelDescription.xml
- Variable names, physical units and start values in modelDescription.xml

Two important features are missing in FMI 2.0. Differential-Algebraic Equation (DAE) systems are not covered. They must be converted to an explicit system of differential equations inside the FMU. This is a performance penalty for optimization solvers that may treat

DAE constraints themselves. This is why FMI is not suited for models with many algebraic constraints, like network models.

Clocked equations are a powerful mechanism to model discrete systems. Unfortunately the FMI event mode and the ModelStructure do not cover discrete-time states resulting from clocked equations. This makes the treatment of discrete-time models clumsy.

4 OpenModelica

OpenModelica offers an open development process, including published nightly tests and a public discussion panel. This eliminates hidden problems. It ensures a high quality and makes OpenModelica well suited for code export to control applications that shall run 7/24 in possibly safety critical environments.

Moreover OpenModelica has outstanding support for localization. The GUI is delivered with 9 translations and the development environment fully supports UTF-8 for localized doc strings. This broadens the range of possible applications beyond nerds.

OpenModelica enables extensions by third parties for particular needs, like model-based control and real-time applications. Figure 1 gives an overview of the main modules and the data flow in the OpenModelica compiler (OpenModelica, 2014).

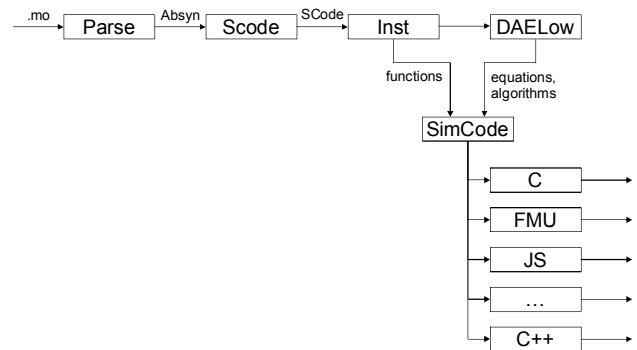


Figure 1: Overview of OpenModelica compiler

The parser generates an abstract syntax tree (Absyn), which is converted to the simplified intermediate code (SCode) and instantiated to a Differential Algebraic Equation system (DAE). The backend DAELow simplifies the equations and algorithms, applies DAE index reduction and brings the equations to the Block Lower Triangular (BLT) form.

The SimCode module applies a template mechanism to generate code for a specific target. There exist multiple code generators, covering the C runtime, FMU export, JavaScript and more. The C++ runtime was developed with particular requirements of real-time simulation in mind (Worschech and Mikelsons, 2012).

4.1 Common Sub-expression Elimination

Common sub-expressions, like a function called multiple times for the same arguments, may result from object-oriented libraries and minimal connector interfaces.

Take Modelica.Fluid as example. Only pressure p and specific enthalpy h appear in fluid connectors, besides fluid composition. Connected components may call the same function to obtain the temperature $T(p, h)$ on each side of the connection.

A Modelica tool should eliminate common sub-expressions such that they are evaluated only once.

4.2 C++ runtime

Most Modelica tools translate models to C code and simulate them with a runtime written in C as well. OpenModelica offers the possibility to generate simulation code for various languages and runtimes. The default runtime of OpenModelica is also based on C code, but there is a powerful additional runtime written in C++ that can easily be used. By comparing these two runtimes, it can be noticed that the C++ code leads to a higher compilation time, but gives a better runtime performance. Besides that, the C code is less comprehensible and harder to maintain.

Especially features like memory management and exception handling need to be implemented manually in C, messing up the code. Similar code has to be written multiple times, for example to implement arrays of different data types like double, int, bool, string, and records.

C++ addresses many of these issues. It not only has built-in exception handling and object destructors for automated memory management, it also offers templates for an advanced reuse of written code. C++ compilers can instantiate one and the same template multiple times for different data types. Appropriate use of this feature also increases type safety. It may even shift effort from model execution to model translation, making the executable code more reliable and efficient.

While reducing source code size and improving type safety, the additional features of C++ lead to longer compilation times. This restricts its use for interactive sessions. Compilation time is less an issue for online applications, where a model is compiled once and then runs endlessly in the real-time control.

4.3 Arrays

“I have never seen a perfect matrix class. In fact, given the wide variety of uses of matrices, it is doubtful whether one could exist” (Stroustrup, 2014).

Modelica models use multi-dimensional matrices and arrays to a large extent. Unrolling these arrays during

model translation is not acceptable, as it leads to large code and long translation times. This is why the simulation runtime needs good support for arrays.

The OpenModelica C++ runtime addresses the wide variety of requirements on arrays with polymorphism. Figure 2 shows the different array classes.

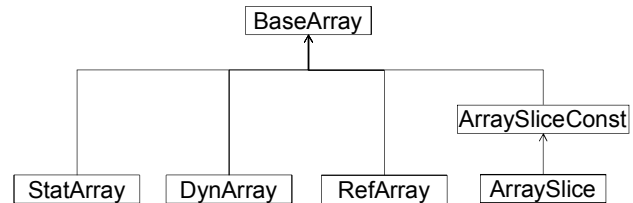


Figure 2: Inheritance diagram of array classes

BaseArray defines a common interface. It is also used as type for array arguments to functions.

StatArray implements that interface with an array of fixed size, known at compilation time. The array data is stored inside the object itself, in order to improve runtime performance.

DynArray can be resized at runtime. It stores array data in dynamically allocated memory. It is typically used in functions that operate on arrays of variable size, like `input Real[:, :] A`.

RefArray is a static array of pointers to simulation variables. This way the array elements may be distributed to optimize performance (see section 4.4 below).

ArraySlice holds a reference to a BaseArray and gives access to a sub-array without necessarily copying the data. It directly maps the Modelica slice syntax to C++, e.g. for `A[1:3, :]`.

ArraySliceConst implements a subset of the functionality of ArraySlice, giving read access only. This is needed for slices of const arrays.

4.4 Performance optimizations with RefArray

The RefArray-type is a simple data structure that can help to improve the performance of large model simulations. Most Modelica tools generate simulation code that stores the array elements consecutive in memory. A solver algorithm is used to calculate all equations respectively equation systems of the model. The execution order of these equations is defined during model translation in the Modelica compiler itself. Because the array variables of the model are often unrolled in the backend of the Modelica Compiler, the single array elements are not solved in the same equation or equation system, but interspersed throughout different equations. This can lead to caching problems, because modern CPU cache memories follow the principle of locality (Denning, 2005). The hardware will automatically prefetch values that are stored besides the memory locations that were

used in the last instructions. Thus, the required variables should be stored as dense as possible in memory in order to reduce cache misses and enable efficient computation. With the RefArray-type it is possible to store all variables that are required to solve one equation as dense as possible in memory, because all array elements are references that can be distributed arbitrarily.

4.5 Array storage order

The storage order of multi-dimensional arrays (row major or column major) is generally arbitrary.

Most Modelica runtimes follow the common C convention of row major order. On the other hand most external functions require column major order (in particular LAPACK called from Modelica.Math.Matrices). Thus transposition operations are necessary on each external function call (e.g. when a matrix A shall be inverted with $\text{inv}(A)$ and inv calls `LAPACK.dgetri`).

The C++ runtime stores array data in column major order. External functions can be called without overhead this way. The C++ array implementation hides the storage layout behind its interface. For example the second element of the first row is accessed in Modelica with `A[1,2]` and in the C++ runtime with `A(1,2)`, independent of the internal storage order.

4.6 Mapping of base types

Table 1: Mapping of Modelica base types to C++ types

Modelica type	C++ type
Real	double
Integer	int
Boolean	bool
String	std::string

Table 1 shows the mapping of Modelica base types to C++. It is assumed that the C++ compiler maps its default base types to the most appropriate and efficient binary representations of the respective hardware platform.

It might be considered a drawback that `int` will typically be 32 bit even on a 64 bit architecture. On the other hand the mapping to standard base types improves platform independence and it simplifies the integration with other software packages, like numerical FORTRAN routines. Moreover note that the IEEE 754 representation of `double` has 64 bits and can treat exact integers with up to 53 bits in a platform independent way.

The C++ language is paired with a powerful standard library. The `std::string` gives the ability to treat a

character string like a regular base type. This simplifies the coverage of strings by the C++ runtime.

4.7 Real-time behavior

There are some critical facts that have to be considered for real-time simulations. First of all, most of the program execution time should be spent in user mode and not in kernel mode, to prevent context switches that are expensive and can bloat the simulation time. For the generated simulation code, the most critical part that leads to these kind of context switches is memory allocation. Therefore, the C++ simulation runtime allocates the required memory during initialization and frees it after the simulation run. One exception was described in section 4.3 with the `DynArray` type, which is rarely used in the evaluated simulation models and thus not a problem for the real time behavior.

Secondly the time integration solver and its event handling are important for real-time simulation. The C++ simulation runtime offers clear interfaces to change the solver and adapt it for real-time criteria. No further details are given here because this was already described in (Worschech and Mikelsons, 2012).

Finally, it should be noticed that C++ object oriented code can lead to small code size and small binaries, which is important for real-time simulation as well.

5 Application example

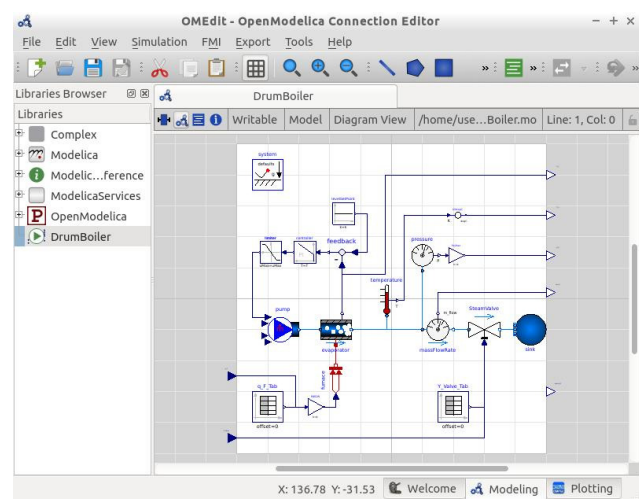


Figure 3: DrumBoiler model in OMEdit

The DrumBoiler example was first introduced in (Franke et al, 2003). Meanwhile it has been added to the Modelica Standard Library and was re-formulated using regular Modelica.Media and Modelica.Fluid. Extended versions of the model, including also once-through boilers, superheaters, reheaters and turbine stages, have been installed in many steam power plants worldwide, optimizing boiler startup control and plant performance.

The distribution of the optimization solver HQP contains two optimization examples for the basic DrumBoiler model: a steady-state set point optimization and a dynamic start-up optimization. The FMI based examples obtain Jacobians with finite differences so far.

Figure 3 shows the model in the OpenModelica editor OMEdit. The model has three states: drum pressure, liquid water level and integrated error of the feed water controller. The model contains discrete events for flow reversal and control limits. These events are irrelevant here because they are not triggered during the startup optimization.

The objective of the startup optimization is to reach given set points for steam pressure p and flow rate q_m :

$$J = \int_{t=t_0}^{t_f} w^T \left\{ \begin{array}{l} [p_S(t) - p_{ref}]^2 \\ [q_{m,S}(t) - q_{m,ref}]^2 \\ \left[\frac{dq_F(t)}{dt} \right]^2 \end{array} \right\} dt \rightarrow \min_{u(t)}$$

subject to bounds on the control $u = (q_F, Y_{Valve})$, i.e. fuel flow rate and valve position:

$$0 \leq q_F \leq 500 \text{ MW}$$

$$0 \leq Y_{Valve} \leq 1$$

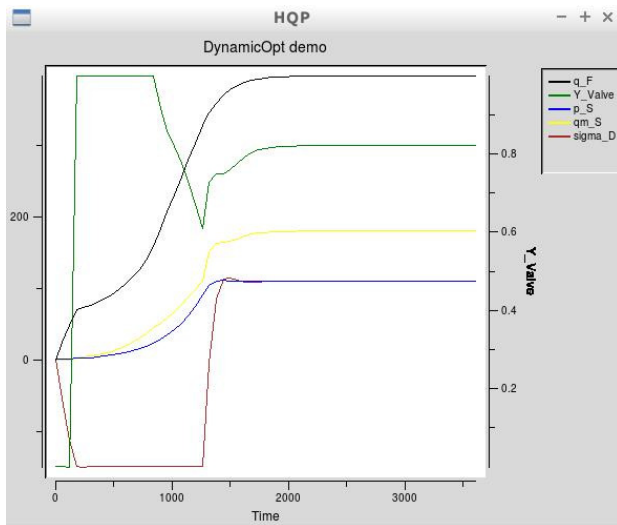


Figure 4: Results of the startup optimization example
rate of change bounds:

$$-24 \text{ MW/min} \leq \frac{dq_F}{dt} \leq 24 \text{ MW/min}$$

as well as an output constraint on thermal and membrane stress that is a function of drum temperature differences and pressure:

$$-150 \frac{N}{mm^2} \leq \sigma_{Drum} \leq 150 \frac{N}{mm^2}$$

$$\sigma_{Drum} = 10^{-3} \frac{dT_{Drum}}{dt} + 10^{-5} p_{Drum}$$

The time horizon spans over one hour. It is split into 60 equally spaced intervals with a length of 60 seconds. The control trajectories are parameterized piecewise linear. The continuous-time differential equations are solved in each interval with two fixed steps of 30 seconds applying the Implicit Midpoint Rule (IMP).

Figure 4 shows optimization results. The fuel flow rate and the valve position are controlled such that the constraint on thermal stress is fully exploited while approaching the target operating point. The optimized control trajectories consisting of 60 linear line segments appear smooth in the plot.

5.1 Runtime performance

Table 2: CPU times obtained in a VirtualBox running Linux jessie 3.16, x86_64 on a MacBook Pro Late 2013 with 2.4 GHz Intel Core i5. The gcc version is 4.9.2.

Modelica Tool for FMU export	CPU time with gcc flag		
	-O0	-O2	-Ofast
OpenModelica 1.9.3	16.6 s	15.5 s	13.5 s
OpenModelica 1.9.3 +cseCall	6.0 s	5.5 s	5.2 s
Dymola 2015FD01	3.4 s	1.7 s	1.3 s
OpenModelica 1.9.3 +simCodeTarget=C++	5.6 s	1.9 s	1.0 s
OpenModelica 1.9.3 +simCodeTarget=C++ +cseCall	2.7 s	1.0 s	0.6 s

Table 2 lists CPU times obtained for different FMUs of the same model. The results for the regular OpenModelica C runtime show that the elimination of common sub-expressions with the flag +cseCall is crucial for the fluid model. The Dymola results serve as reference.

The C++ runtime is selected with the flag +simCodeTarget=C++. It uses the same SimCode input as the C runtime and generates C++ code from it. The C++ runtime has its own FMI implementation.

The gcc optimization flag has only minor impact on the OpenModelica C runtime. An improvement by a factor of 2-3 is seen for Dymola C code. The OpenModelica C++ runtime shows a speedup by a factor of 4-6 with compiler optimization. This huge improvement underlines that the higher level expressiveness of C++ is actually exploited by modern compilers.

The CPU times reported here are the average of 10 runs. The deviations between different runs as well as the impact of the virtualization environment on the CPU times are minor. This is important because repeated runs in virtual production environments mark a major use case.

5.2 Reproduction of results

The results reported here can be reproduced on a Posix compliant machine with reasonable development tools installed (on a Debian based system these are the packages: `git`, `gcc`, `g++`, `tcl-dev`). One might invoke the commands:

```
$ git clone https://github.com/omuses/hqp.git
$ cd hqp
$ ./configure
$ make
$ cd odc
$ ./run drumboiler
```

The last command evaluates `drumboiler.tcl`. This file contains the optimization specification and all solver settings. They are equal, no matter how the FMU was generated.

The Tcl script calls the OpenModelica compiler `omc` with default settings to generate an FMU from the simulation model `DrumBoiler.mo`. Alternatively the FMU can be generated separately and copied to the `odc` directory before running the optimization.

Note that the first run with a newly generated FMU contains an unzip operation. Each run parses the file `modelDescription.xml`. The impact of the XML parser vanishes if multiple runs are performed in one process. The average time of ten runs in one process is obtained with:

```
$ ./odc
% time {source drumboiler.tcl} 10
```

6 Conclusions

FMI 2.0 for model exchange provides an efficient interface for hooking simulation models to optimization solvers and running model-based control applications. The approach discussed in this paper offers several advantages over alternative optimization approaches. The standardized FMI hides details of particular modeling tools or components thereof, enabling innovations without comprising a whole tool chain. Several Modelica tools support FMI.

OpenModelica offers a modular environment that can be customized and, thanks to the open source setup, further developed for particular needs.

The default C runtime is a good compromise between fast compilation speeds and high runtime performance. It is suited for interactive modeling and simulation sessions. It has limitations for model-based control applications though. Especially the garbage collection can produce issues.

The C++ runtime is particularly developed for real-time simulation. It exploits object destructors for determinis-

tic memory management. C++ has a rich syntax to express programming concepts on a high level. This not only improves readability by humans, it also enables more code optimization by C++ compilers. Exploiting templates, the amount of manually written code is minimized and type safety is increased. For instance an array class only needs to be implemented once for arbitrary types of array elements. This boosts development efficiency and reduces the probability of bugs.

Some missing features were added to the C++ runtime throughout the work reported here. In particular the existing FMI 1.0 export was upgraded to FMI 2.0 and some issues were solved in the OpenModelica backend for FMI export. The array implementation was enhanced, an external FORTRAN interface was added. The array storage order was changed from row major to column major to minimize the overhead when calling external functions, like LAPACK functions from `Modelica.Math.Matrices`.

The OpenModelica development process with nightly tests and public issue tracking helped significantly. It provides immediate feedback on the progress made and possible negative side effects.

As a result the C++ runtime is applicable to model-based control using FMI 2.0 for model exchange along with the widely used optimization solver HQP. A speedup of up to 8 is seen with `gcc` optimization flags. An example shows an FMU exported with the C++ runtime performing significantly faster than the FMU exported with the C runtime or with Dymola.

The price to pay with C++ is longer compilation times. This is less an issue for online control applications, where a model is compiled once and then runs endlessly in the real-time control.

Another possible drawback of C++ is stronger coupling between compilation modules, as required for improved type safety, performance, and exception handling. These things are hidden behind FMI.

FMI needs to be further developed towards supporting DAE constraints, e.g. arising from network models, and discrete states arising from clocked equations. The OpenModelica C++ runtime offers a promising basis for this future work.

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.

References

- P. J. Denning: The locality principle, *Communications of the ACM - Designing for the mobile device*, July 2005.
- R. Franke, E. Arnold: Applying new numerical algorithms to the solution of discrete-time optimal control problems. In: *Computer Intensive Methods in Control and Signal Processing: The Curse of Dimensionality*, Birkhäuser, Basel, 1997.
- R. Franke, M. Rode, K. Krüger: On-line Optimization of Drum Boiler Startup, 3rd International Modelica Conference, 2003. https://www.modelica.org/events/Conference2003/papers/h29_Franke.pdf
- R. Franke, L. Vogelbacher. Nonlinear model predictive control for cost optimal startup of steam power plants. *at – Automatisierungstechnik*, 54(12):630–637, 2006.
- R. Franke, B.S. Babji, M. Antoine, A. Isaksson: Model-based online applications in the ABB Dynamic Optimization framework, 6th International Modelica Conference, Bielefeld, March 3-4, 2008.
- R. Franke, S. Saliba, A. Frick: Virtual Power Plants for Smart Markets, *PowerGen Europe*, Cologne, June 2014.
- Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0, July 2014.
- E. Lazutkin, A. Geletu, S. Hopfgarten, P. Li: Modified Multiple Shooting Combined with Collocation Method in JModelica.org with Symbolic Calculations, *Proceedings of the 10th International Modelica Conference* March 10-12, 2014, Lund, Sweden. <http://www.ep.liu.se/ecp/096/104/ecp14096104.pdf>
- F. Magnusson, K. Berntorp, B. Olofsson, J. Åkesson: Symbolic Transformations of Dynamic Optimization Problems, *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, March 10-12, 2014.
- Z.K. Nagy, B. Mahn, R. Franke, F. Allgöwer. Evaluation study of an efficient output feedback nonlinear model predictive control for temperature tracking in an industrial batch reactor. *Control Engineering Practice*, 15(7):839 – 850, 2007.
- J. Neupert, E. Arnold, O. Sawodny, and K. Schneider: Tracking and anti-sway control for boom cranes. *Control Engineering Practice*, 18(1):31–44, 2010.
- OpenModelica System Documentation, February 2014.
- A. Pfeiffer: Optimization Library for Interactive Multi-Criteria Optimization Tasks, *Proceedings of the 9th International Modelica Conference*, September 3-5, 2012, Munich, Germany. <http://www.ep.liu.se/ecp/076/068/ecp12076068.pdf>
- V. Ruge, W. Braun, B. Bachmann: Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL-C, *Proceedings of the 10th International Modelica Conference*, March 10-12, 2014, Lund, Sweden.
- B. Stroustrup: *The C++ Programming Language*, Fourth Edition, Addison-Wesley Pearson Education Inc., 2014.
- J. Wagenpfeil, E. Arnold, H. Linke, O. Sawodny: Modeling and optimized water management of artificial inland waterway systems. *Journal of Hydroinformatics*, 15(2):348–365, 2013.
- N. Worschech, L. Mikelsons: A Toolchain for Real-Time Simulation using the OpenModelica Compiler, 9th International Modelica Conference, Munich, 2012. <http://www.ep.liu.se/ecp/076/086/ecp12076086.pdf>
- D. Zimmer, M. Otter, H. Elmqvist, G. Kurzbach: Custom Annotations: Handling Meta-Information in Modelica, *Proceedings of the 10th International Modelica Conference*, March 10-12, 2014, Lund, Sweden. https://www.modelica.org/events/modelica2014/proceedings/html/submissions/ECP14096173_ZimmerOtterElmqvistKurzbach.pdf
- J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, H. Tummescheit. Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problems. *Computers and Chemical Engineering*, 34(11):1737–1749, November 2010.

Nonlinear Dynamic Inversion Control for Wind Turbine Load Mitigation based on Wind Speed Measurement

Matthias J. Reiner¹ Dirk Zimmer¹

¹Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
{Matthias.Reiner, Dirk.Zimmer}@DLR.de

Abstract

The design of an advanced controller for wind turbine load mitigation is presented. The controller is based on Nonlinear Dynamic Inversion control methods combined with Pseudo Control Hedging to account for the actuator limits and a two degree of freedom control system for the collective pitch control of the rotor blades. The controller uses wind speed measurement information to adjust to wind gust load. A newly developed wind turbine system dynamics library in the Modelica language is used to model an elastic wind turbine for a simulation study of the controller. The simulation results show a large reduction of the gust load on the wind turbine using the proposed controller.

Keywords: Elastic wind turbine modeling; nonlinear dynamic inversion; pseudo control hedging; optimization

1 Introduction

Wind energy has become an important energy source with worldwide growing capacities. Advanced control design can help to improve the energy generation and extend turbine lifetime.

While the nominal control of wind turbine is already well handled by the state of the art, and only minor improvements can be expected for the ideal case with smooth wind speed, there is still much potential in the field of load reduction. Especially under gust load conditions, the load on the flexible structure of the wind turbines can be reduced using advanced control methods.

One important aspect in this regard is the technological advance in wind speed measurement. Especially turbine-mounted light detection and ranging (LIDAR) based wind speed sensors have become much more affordable and accurate. The measurement of the wind speed opens a wide range of control methods. Gust load can be substantial and induce strong vibrations of the wind turbine tower, which can lead to a lifetime reduction.

In recent years, many different advanced control approaches have been proposed for the control of



Figure 1. Visualization of a Modelica Wind Turbine model using the DLR SimVis Library (Bellmann, 2009).

wind turbines that are based on wind speed measurements (Dunne et al., 2011). Especially methods based on Model Predictive Control (MPC) strategies and feed-forward disturbance compensation methods show promising results, e.g. (Schlipf et al., 2010; Wang and Johnson, 2011; Koerber and King, 2013). This simulation study focuses on an approach based on Nonlinear Dynamic Inversion (NDI) (Slotine and Li, 1991) combined with Pseudo Control Hedging (PCH) (Johnson and Calise, 2000). Similar controller structures are known from the field of aerospace control. Although not directly comparable, there are many similarities: in both cases, the main source for the nonlinearity results from the changing wind speed (e.g. airspeed) and resulting

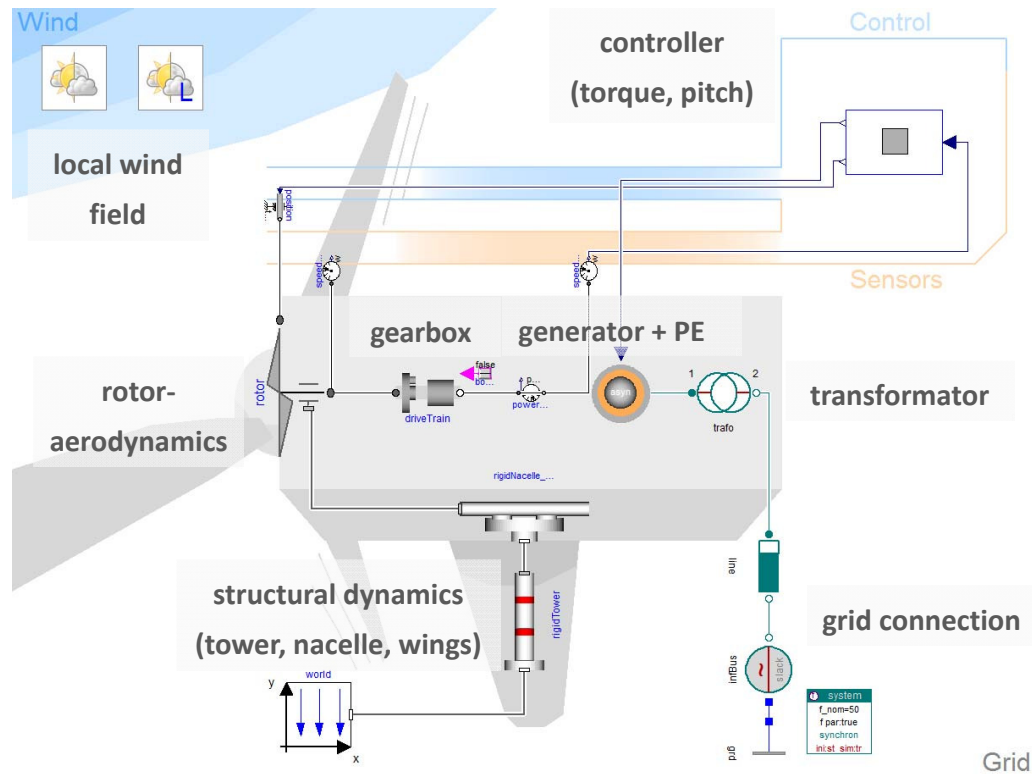


Figure 2. Top level Modelica model diagram of a typical wind turbine architecture.

aerodynamics. Actuator limits play a very important role (rates and absolute limits) and the excitation of the flexible structure has also to be considered in both cases during gust load situations.

Since good results have been achieved in the field of aerospace control using NDI with PCH, it is worthwhile to study its adaption and application for elastic wind turbines. One important aspect for the control of wind turbines are the dynamic ranges of the actuators. While the turbine generator can be controlled very fast to react to load changes, the pitch actuators of wind turbines are usually relatively slow. NDI has also recently been investigated for the pitch control of wind turbines (Geng et al., 2014) but we focus on the generator control, where a faster response can be achieved and use a two degree of freedom system for the pitch control. The actuator to turn the wind turbine into the wind direction (yaw axis) is usually the slowest actuator and therefore not relevant for load mitigation control and will not be considered here.

Section 2 describes the approach used for modeling the elastic wind turbine using the Modelica language (Modelica Association, 2010). Based on this nonlinear model the NDI & PCH based controller is described in section 3 using Modelica and the Functional Mock-up Interface technology (FMI) (development group, 2014). Simulation results and comparisons to a conventional scheduled controller are given in section 4. A conclusion and outlook is given in section 5.

2 Modeling of elastic wind turbines

Modern wind turbines represent multi-domain systems. Hence, to describe the dynamic behavior of a complete turbine, expertise from different areas is required: wind field analysis is needed to describe the environment; rotor-aerodynamics models the transformation of wind energy into kinetic energy; this kinetic energy is then turned into electric energy by the means of gears, electric machines and power system converters; also the flexibility of the tower structure and the blades needs to be taken into account.

Fig. 2 illustrates how all these components are ultimately assembled for a complete turbine model within Modelica. It represents one specific example of a wind turbine that is based on the 5MW reference turbine from NREL (Jonkman et al., 2009). The electric parts of the turbine (generator, converter, grid) have been adapted to the specifics of the European market.

Given any task, the modeler will need to adapt this turbine model to her or his specific needs. For instance, modeling the integration of a turbine into electric grid requires more detailed modelling of the generators and converters, whereas for the task of this paper, the generator can simply be represented as a controlled source of limited torque. It is thus necessary that each part of the turbine can be modeled with the appropriate level of complexity and detail. To ensure this flexibility, the modeling work is supported by a DLR library dedicated to

wind turbines. In this library, DLR offers its combined modeling know-how in the field of drive-train (Tobolar et al., 2007), aerodynamics (Looye, 2008), structural dynamics (Heckmann et al., 2006), and visualization (Bellmann, 2009) (see Fig. 1) in order to offer a complete system dynamics library for wind turbines. The following sections describe those parts of the library which are relevant for the control task at hand. These are: the modeling of the aerodynamics, the structural dynamics and the design of a standard controller.

2.1 Wind turbine aerodynamics

The evident key-component of a wind turbine is its rotor. The most-straight forward approach is to regard this component as one single entity that transforms wind energy into kinetic energy by a coefficient of efficiency C_p . This coefficient is typically defined as a function of the tip-speed ratio λ , expressing the relation between wind speed and the tip speed of the blades. If we take the collective pitch of the blades into account, we get a two-dimensional function, where both input parameters are subject to control laws: the pitch angle is controlled by the pitch actuators and the tip speed can be regulated by the power-off take of the generator. Hence this basic model of a rotor is well applicable for classic control tasks.

For more advanced tasks, the rotor is decomposed into its blades and the blades are decomposed into blade elements. Each such element then has its own airfoil polar that describes the aerodynamic forces for lift and drag dependent on the angle of attack within the induced wind field. The induced wind field is a superposition of the global wind in the vicinity of the rotor field and the wind that is induced by the rotor itself. For a certain induced wind, the momentum needed to change the wind velocity and the momentums emanated from the aerodynamic forces of the blade element are equal. This equilibrium can be described by a nonlinear system of equations and forms the basis of the well-established Blade-Element-Momentum (BEMT) Method (Hansen, 2008). It is suitable for wind turbines without strong cross-winds or significant local turbulences. It can be used to take the influence of wind shear into account and to design individual pitch control systems. In combination with flexible blades and towers, good estimation of the structural loads can be performed.

For the method's implementation in Modelica, a model of a local blade element has been created. In interaction with a global outer model for the rotor plane, this component determines the local aerodynamic forces. The individual blade elements can then be connected (rigidly or by a flexible body) to form a complete blade. Typically, three such blades then form the rotor.

2.2 Structural dynamics of tower and blade

Modern, large-scale wind turbines have rotors of more than 100 meters of diameter. The flexibility of such a large structure is an integral part of its design. Most important are the structural dynamics of those components which are most exposed to the wind: the tower and the blades. The flexibility of the nacelle, especially of the mounting of its devices and bearings, is currently not taken into account but the elasticity of the nacelle mounting on its tower is taken into account around the yaw angle.

For the modeling of these components, the DLR FlexibleBodies Library (Heckmann et al., 2006) is used. Using this library, the components can be modeled using standard connectors of the Modelica Standard library. The blade-elements of the rotor aerodynamics can hence be connected to a flexible blade as well as to a rigid blade.

The components of the FlexibleBodies library are based on a modal approach. Structural information of the blade and tower stiffness, as presented in (Jonkman et al., 2009) can be incorporated into the model by a SID file (Heckmann et al., 2006). Nonlinear effects such as the increased stiffness due to centrifugal stretching can be taken into account. The performance characteristics are comparable to the approach presented in (Thomas et al., 2014).

2.3 Complete turbine model

For the control task of this paper, the aerodynamics and structural elasticity are the key components of the wind turbine. Nevertheless, more components are needed to form a complete model. This is also shown in Fig. 2. Wind models are needed that prescribe the regional wind conditions but also model local effects like height dependent wind shear and the tower dam effect. Components for gearbox, emergency brake, generator and power electronics form the complete drive train of the turbine model. Many different designs for such drive trains exist in current wind turbines and the presented diagram just shows one possible setup.

Finally, the controller of the turbine model is depicted in Fig. 2 with its signals controlling motor torque and pitch angle. One possible and advanced design of such a controller is presented in the following chapters.

3 Controller design

The nonlinear characteristics of the wind turbine are considered in our control design by using an inverse model of the wind turbine as part of the controller. The approach used here is Nonlinear Dynamic Inversion (NDI) for the control of the generator torque together with Pseudo Control Hedging to handle actuator limits.

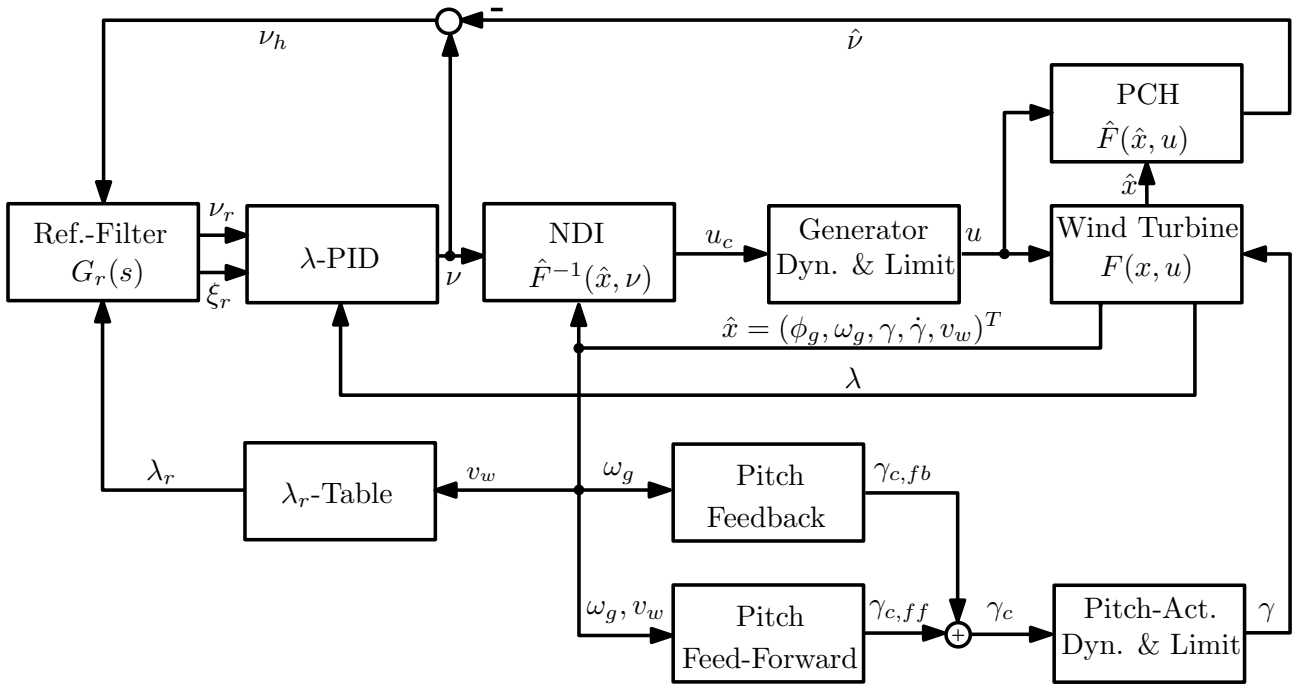


Figure 3. Overview of the combined NDI & PCH control system for the generator torque in addition to a two degree of freedom pitch controller. The connections show the data flow between the different components.

For our controller design, it is assumed that high accuracy measurement of the wind speed at the turbine is available. This could be based on LIDAR, e.g. (Simley et al., 2012) or similar measurement methods in combination with additional estimators or filters.

The notation and implementation used here is similar to (Looye, 2001; Holzapfel, 2004; Lombaerts et al., 2012) from the field of aerospace control.

As an extension for the NDI control of the generator torque, a feed-forward controller for the pitch control is used. The feedback controller design for the pitch control is based on the reference controller from (Jonkman et al., 2009) that will also be used for comparison of the controller performance in Sec. 4. The reference pitch controller from (Jonkman et al., 2009) is a scheduled (on the generator speed) PI controller. Under nominal conditions the controller already works very well, but can be improved by a wind speed feed-forward controller.

The feed-forward controller for the pitch actuator consists of different elements: the main part is a linear interpolated table which contains a function $\gamma_{tab}(v_w)$ of optimal pitch actuator angles γ depending on the measured wind speed. The optimal values are the steady-state results for a simulation using only the reference pitch controller and constant wind speed. Since the pitch actors are relatively slow, they should only be used when the generator is close to its limit. For this reason a switch is implemented in the feed-forward controller that only enables the controller close to the lower or upper limit of the generator speed ($\omega_{g,low}$ and $\omega_{g,high}$). A hysteresis loop is implemented to avoid jittering of the switch close to the limit. When active, the output of $\gamma_{tab}(v_w)$ is

used as input for a low-pass Bessel filter $G_{bes}(s)$ (Laplace variable s) and a rate limiter (limit $\dot{\gamma}_{c,max}$) to ensure that the pitch actuators can follow the dynamics of the feed-forward controller.

$$\varkappa_{on} = \omega_g > \omega_{g,high} \vee \text{pre}(\varkappa_{on}) \wedge \omega_g \geq \omega_{g,low} \quad (1a)$$

$$\gamma_{c,uf} = \begin{cases} \gamma_{tab}(v_w), & \text{if } \varkappa_{on} = 1, \\ 0, & \text{if } \varkappa_{on} = 0. \end{cases} \quad (1b)$$

$$\gamma_{c,fi} = G_{bes}(s)\gamma_{c,uf} \quad (1c)$$

$$\dot{\gamma}_{c,ff} = \min \left(\max \left(\frac{\gamma_{c,fi} - \gamma_{c,ff}}{T_r}, -\dot{\gamma}_{c,max} \right), \dot{\gamma}_{c,max} \right) \quad (1d)$$

$$\gamma_{c,ff} = \int \dot{\gamma}_{c,ff} dt \quad (1e)$$

Eq. 1 shows the resulting equations and logic for the feed-forward controller. The feed-forward controller is combined with the scheduled PI pitch controller from (Jonkman et al., 2009) to form a two degree of freedom control system.

We assume the general nonlinear system description of the wind turbine in the form of Eq. (2) where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^k$ is the input vector, $y \in \mathbb{R}^l$ is the vector of outputs that are controlled, $p \in \mathbb{R}^{n_p}$ are the parameters and known inputs of the system and $z \in \mathbb{R}^m$ contains any other outputs of the system. Any other control inputs are considered as known parameters ($\in p$). The nonlinear model is our implementation based on the 5-MW reference wind turbine from (Jonkman et al., 2009),

using the model library described in Sec. 2.

$$\dot{x} = f(x, p) + g(x, p)u \quad (2a)$$

$$y = h(x, p) \quad (2b)$$

$$z = h_0(x, p) \quad (2c)$$

The inverse system can be generated using Lie-derivatives of the output h along f and g .

$$\dot{y} = L_f h(x, p) + L_g h(x, p)u \quad (3)$$

Assuming that $L_g(x, p)$ is invertible and all outputs have the relative order 1 with respect to one of the inputs, the following control law can be constructed:

$$u_c = (L_g h(\hat{x}, p))^{-1} (\dot{y}_d - L_f h(\hat{x}, p)) \quad (4)$$

In Eq. (4) $y_d \in \mathbb{R}^l$ represents the demand rates and \hat{x} represents a subset and estimation of the states x of the original system.

$$\hat{x} = (\phi_g, \omega_g, \gamma, \dot{\gamma}, v_w)^T \quad (5)$$

In the following, we will use a shortened notation where $F(x, u)$ is used for Eq. (2), $\hat{F}(\hat{x}, u)$ for a differentiable approximation of $F(x, u)$ for which all assumptions are valid. For $\hat{F}(\hat{x}, u)$, the flexible dynamics are neglected and the tables for the nonlinear aerodynamics are replaced by differentiable B-splines. For the modified system only a subset \hat{x} of the states of the original system x are used. The subset of the states \hat{x} in Eq. (5) consist of the generator angle ϕ_g and angular velocity ω_g , the pitch actuator angle γ and angular velocity $\dot{\gamma}$ as well as mean wind speed at the rotor blades v_w .

The inverse system based on $\hat{F}(\hat{x}, u)$ and Eq. (4) is called $F^{-1}(\hat{x}, v)$ using the virtual control input $v = \dot{y}$. Using $\hat{F}^{-1}(\hat{x}, v)$ to generate the input u_c for \hat{F} would lead in the ideal case (with $F = \hat{F}$ and all p perfectly known) to an input/output linearization such that the original nonlinear system is transformed to a closed loop system with decoupled linear dynamics:

$$\dot{\hat{x}} = v \quad (6)$$

But even in the case if only $F \approx \hat{F}$ the nonlinearity of the closed loop system can be greatly reduced, such that a linear controller is better able to achieve good performance.

The inverse control law of Eq. (4) can be generated automatically from a modified Modelica model of the wind turbine using the automatic differentiation and index reduction (Mattsson and Söderlind, 1993) features of Dymola (Otter et al., 1996). The inverse model is then converted to an FMI for which the states of the inverse system are transformed to additional inputs. An outer control loop is used in combination with $\hat{F}^{-1}(\hat{x}, v)$ to control the resulting dynamic of Eq. (6) and to damp effects of modeling errors and parameter uncertainties.

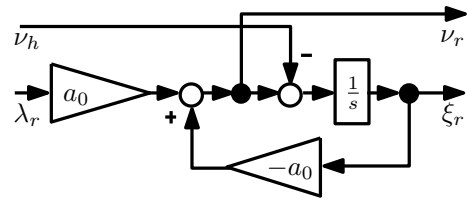


Figure 4. First order reference filter $G_r(s)$ for the PCH.

Since the actuators of wind turbines are limited and also have a dynamic behavior additional measures are necessary. An approach similar to (Holzapfel, 2004) and (Lombaerts et al., 2012), from the field of aerospace, is used here. It consists of a reference filter $G_r(s)$ (Fig. 4) and a model of the system $\hat{F}(\hat{x}, u)$ in combination with a PID-feedback controller in the outer loop. An overview of the controller setup is shown in Fig. 3. The reference filter is used to modify the controller demand such that the actuator limits are maintained. To achieve this, a parallel model of the plant $\hat{F}(\hat{x}, u)$ is used. The input for the parallel model are a subset of the measured states \hat{x} , defined in Eq. (5), of the wind turbine F . The outputs of the reference filter are the set point for the PID controller ξ_r and a term v_r that is directly added to the output v of the PID controller. The reference filter is parameterized as a critical damped filter with cut off frequency f_r (filter parameter $a_0 = 2\pi f_r$).

It is assumed that all these quantities are measurable or obtainable using an observer. For \hat{F} the flexible dynamics of the wind turbine are neglected, i.e. no elasticity of the powertrain and tower is considered in $\hat{F}^{-1}(\hat{x}, v)$. This means ϕ_g and ω_g can also be directly used to calculate the blade speed, except for a constant factor for the gear ratio I_{gen} and tip speed ratio $\lambda = \omega_g R / v_w$ (rotor radius R).

There are multiple reasons for approximation: If the elasticity of powertrain and especially the elasticity of the tower would be directly considered for \hat{F}^{-1} the complexity would rise substantially. Although the inversion would still be possible, at least for a flexible powertrain using the automated possibilities of Dymola, higher order derivatives of the inputs and equations of motion for the inverse model would be required (e.g. of λ , v_w and γ). Since these are measurement values, it would make the inverse model very sensitive to noise and uncertainties in these quantities. For the NDI approach also additional measurements of the elastic deformations would then be necessary (since the whole state vector is required for the method) that are often not readily available. In addition the computational time for the control algorithm would rise considerably such that a real time implementation could be problematic. If elastic effects are considered, also the zero-dynamics of the system used for the inversion can be critical. A stable zero-dynamics is necessary for a stable inversion. The zero dynamics can be approximated by the (transmission-) zeros of a linearization of the nonlinear system since the

full nonlinear analysis of the zero dynamics can usually not be solved analytically for complex systems. For the rigid system, all zeros have no positive real part, and therefore do not turn into unstable poles for the inverse system. The reasons mentioned here are likewise main factors why NDI control systems in the aerospace field are usually also based on rigid models, even though the aircraft can show significant elastic deformations.

For \hat{F}^{-1} the inversion is done with respect to $\dot{\lambda}$, so that in Eq. (4) $u_c = \tau_g$ and $\dot{y} = \dot{\lambda}$. The tip speed ratio was chosen for the inversion since the optimal operation point can be found from C_p - λ curves. The model used for the inversion is modified such that the highest derivatives that are required for the inverse model are directly used as inputs and connected to integrator chains of the respective order.

The parallel model used for the PCH algorithm is the same forward model \hat{F}^{-1} used to generate the inverse model \hat{F}^{-1} , but without the added integrator chains and re-defined inputs. The inversion is done using the Modelica/Dymola capabilities (see (Thümmel et al., 2005) for details).

The resulting inverse model of the rigid wind turbine is then exported as an FMI. In the FMI-code, the state of the inverse system is redefined as an input such that no integration of the derivatives is necessary. The resulting inverse system is therefore in the form of Eq. (4). The required Lie-derivatives are solved automatically by Modelica/Dymola. The modified FMI can then be used for a controller implementation directly on target hardware, or can be re-imported into the Modelica/Dymola to simulate the complete control system. To account for the actuator limits, the output of the inverse model u_c is modified by limiter for the absolute value and the rate, using the same limiter equation as in Eq. (1d) for the pitch rate $\dot{\gamma}$. The result is the generator torque u that is used as set point for the wind turbine torque generator and the PCH parallel model \hat{F} (see Fig. 3).

To improve the numerical robustness of the control system and the numeric, the required derivatives of measurement values are calculated using filters. The Modelica approximated derivative blocks (DT1) are used. The time constants of the DT1 elements can be tuned to provide a good compromise between accuracy and robustness against noise as part of the controller synthesis. The DLR multi-case and multi-criteria optimization tool MOPS (Joos et al., 2002) was used to find suitable parameters for the controllers as well as the content of the table, which contains optimal set points λ_r depending on the wind speed v_w using a gridding simulation optimization for different wind speed inputs of the control system. Since it is assumed that v_w is measurable the optimal λ_r can then be found using a linear interpolated table based on v_w (λ_r -Table in Fig. 3).

As will be seen in Sec. 4, the approximate inversion still helps considerably for the reduction of stress caused by elastic deflections because the controller reacts faster

to quickly changing wind speed. This in turn reduces the acceleration peaks at the tower tip and therefore reduces the elastic vibrations.

4 Simulation results

For the simulation study, multiple scenarios of different wind speed trajectories have been analyzed. The nonlinear simulation plant model is based on the 5-MW reference wind turbine from (Jonkman et al., 2009) using modeling components from Sec. 2.

The model consists of a flexible tower (7 flexible modes, using the FlexibleBodies Library), flexible powertrain (modeled as spring damper system), flexible nacelle mounting (yaw axis) and simplified first order generator dynamics. The wind-rotor interaction dynamic is approximated using a tabled pitch correction factor together with a tabled rotor efficiency factor that is dependent on the tip speed ration and pitch correction factor. The controller from (Jonkman et al., 2009) is used as a reference to compare the results (will be referred to as reference controller in the following). The refer-

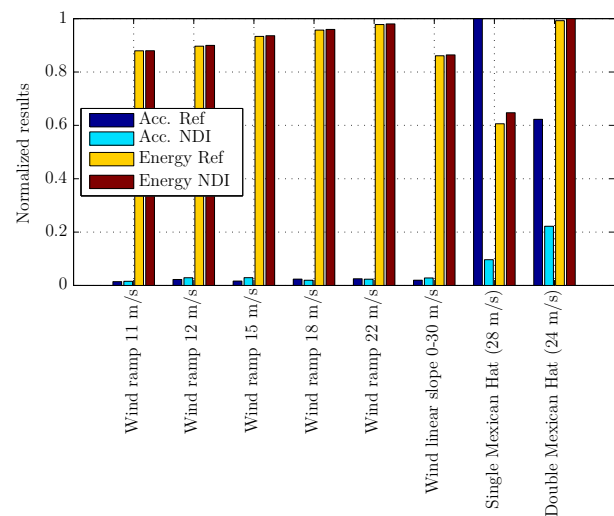


Figure 5. Normalized results of the generated energy of the wind turbine and acceleration at the tower tip for the different simulated scenarios.

ence controller does not use wind measurement information, but works very well under nominal conditions and therefore provides a good benchmark for achievable performance without wind speed measurement. Since the main objective of wind turbines is to generate as much energy as possible, a controller needs to achieve a good compromise between generated energy and load reduction.

It can be assumed that in the long term, load reduction on the tower can lead to an increase of the life time and less required service intervals. This can make the inclusion of a wind speed sensor cost-effective for future wind

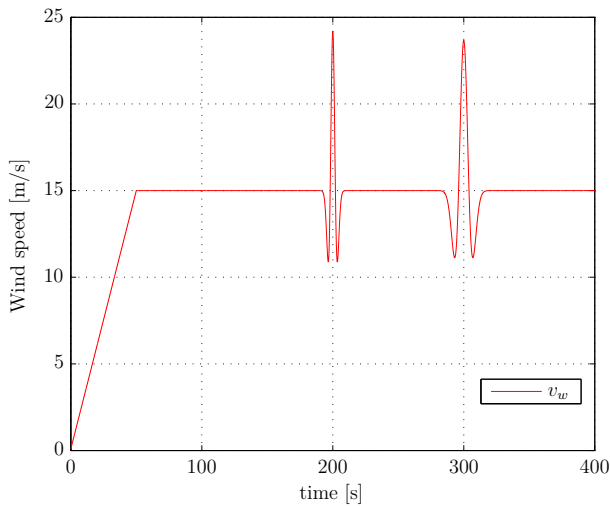


Figure 6. Wind speed for the scenario with a wind ramp and double wavelet disturbances.

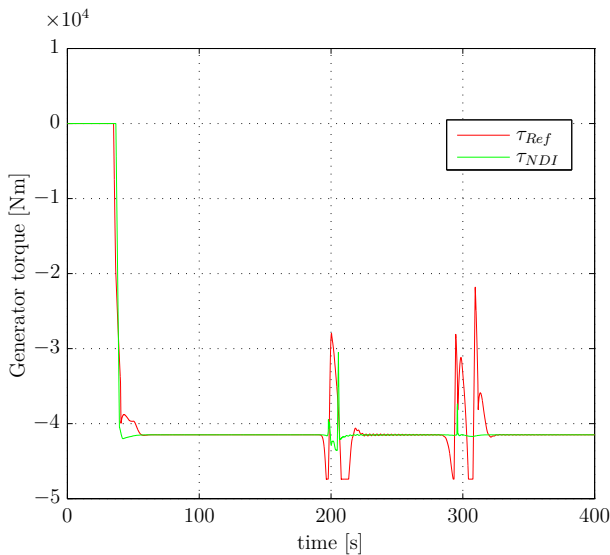


Figure 7. Comparison of the resulting generator torque for the scenario with a wind ramp and double wavelet disturbances.

turbines. To verify the controller performance, eight different scenarios (cases) have been simulated for which the wind speed is varied:

- Wind ramps with different constant end wind speed $v_{w,max}$ (11,12,15,18 and 22 m/s).
- Linear wind speed slope from 0 m/s to 30 m/s.
- Wind ramp with single wavelet disturbance.
- Wind ramp with double wavelet disturbances.

The different cases are used to test different conditions for the control system of the wind turbine. The first five cases with the wind ramps are used to simulate the nominal conditions for the controller with constant wind speed after a startup phase. For these cases no big improvements can be expected compared to the reference

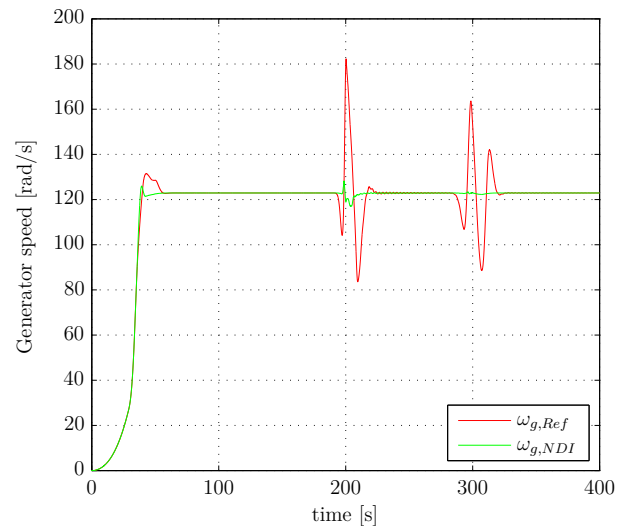


Figure 8. Comparison of the resulting generator speed for the scenario with a wind ramp and double wavelet disturbances.

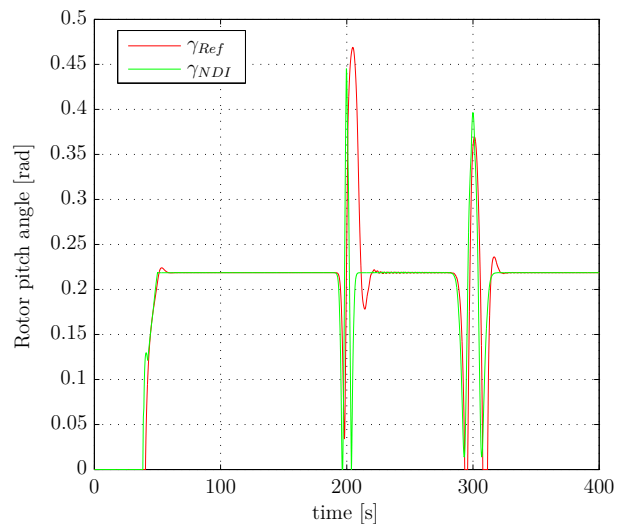


Figure 9. Rotor pitch angle comparison for the scenario with a wind ramp and double wavelet disturbances.

controller. However the cases are important, since it has to be expected that an advanced control system should also be able to generate similar amounts of energy during nominal conditions, and not only focus on load reduction. The linear wind speed slope is used to verify the transition phases for the control law. For example the pitch control and pitch feed-forward control are only active if the generator speed is close to its allowed maximum $\omega_{g,high}$ and also the scheduled PI-Pitch controller has different gains depending on the current pitch angle γ . In addition the set point λ_r for the NDI control loop changes based on the current wind speed v_w . The last two cases are used to simulate the controller behavior under gust load conditions. To simulate the wind gust excitation of the elastic wind turbine tower, Ricker wavelets according to Eq. (7) are used. The wavelets are parameterized to simulate short wind gusts. Because of the form

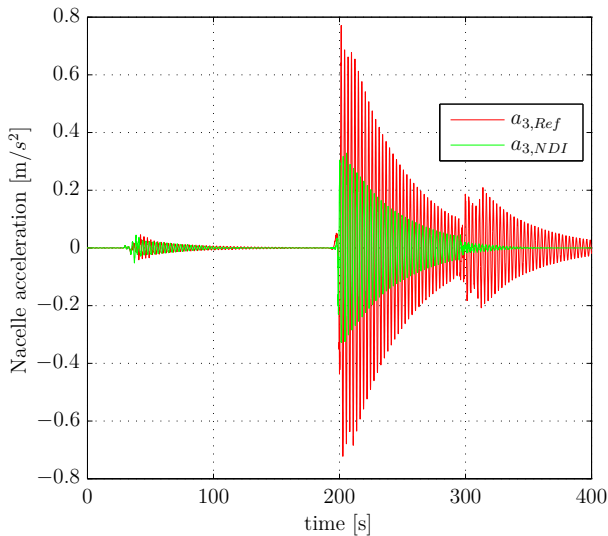


Figure 10. Acceleration vector component parallel to the wind direction at the top of the tower for the double wavelet scenario.

of the resulting shape they are often called Mexican Hat wavelet (see Fig. 6).

$$\psi(t) = \frac{2}{\sqrt{3\sigma\pi^{\frac{1}{4}}}} \left(1 - \frac{t^2}{\sigma^2}\right) e^{-\frac{t^2}{2\sigma^2}} \quad (7)$$

Two criteria have been defined for the simulated scenarios to quantify the generated energy of the turbine and the load of the wind turbine tower in Eq. (8).

$$C_a = \int_0^{400} |a_t|_1 dt \quad (8a)$$

$$C_p = \int_0^{400} \tau_g \omega_g dt \quad (8b)$$

The criteria C_a is a measure for the load and is computed from the integral over the simulation time of 400s of the L1 norm of the acceleration at the tip of the tower a_t . A reduction of a_t is desirable to prolong the turbine's lifetime. The second criteria C_p is the generated energy of the wind turbine during the simulation. An increase in C_p leads to a more efficient turbine. Fig. 5 gives an overview of the results that have been normalized to the maximum values of all cases. As can be seen from the plot, the generated energy is very similar for all cases for the reference controller and the NDI based control system. The NDI controller acts faster what leads to a slight increase in generated energy. However the added benefit from the advanced control law based on the wind measurement can be seen for the resulting acceleration at the tower tip.

The wind gust, simulated by the wavelet, leads to very strong acceleration at the tower tip that can be reduced substantially using the new proposed control system, as can be seen from Fig. 5. In particular, for the case with only one wind peak, the wind gust load can be absorbed directly by the generator using the NDI control law.

Fig. 7 shows the resulting generator torque for this scenario, Fig. 8 the corresponding generator speed, Fig. 6 the wind speed and Fig. 9 the rotor pitch angle. As can be seen from the plots, the controller is able to use the additional information from the measured wind speed to react faster to changes in the wind speed, which results in a much smoother generator speed and generator torque for the NDI controller and additionally reduces the acceleration at the tower tip, as can be seen from Fig. 10. The feed-forward controller for the rotor pitch angle leads to a faster response. The reduced acceleration also lessens the load on the tower structure.

5 Conclusion and future work

The current design of controllers for wind turbines is rather conservative. The results of the last chapters indicate that using advanced control methods such as NDI, significant performance gains can be achieved especially in the field of load mitigation. LIDAR systems, however, are a key technology to enable the described control method. We expect LIDAR systems to become more widespread among new or retrofitted wind turbines due to upcoming reductions in production cost, especially in wind farms where one LIDAR system can be used for multiple turbines.

Modern control methods such as NDI represent model-based approaches. A multi-domain Modelica library for wind turbines with non-causal and hence invertible models forms an optimal basis for the development of suitable NDI controllers.

The simulation results for the NDI & PCH based controller, using advanced capabilities of Modelica and FMI, shows promising results. However in the future it would be necessary to verify the control systems on real wind turbines to analyze the robustness and performance under realistic conditions.

We hope that the simulation-based analysis can help to motivate the use of advanced control systems and sensors in the future.

6 Acknowledgment

The development of Modelica models for wind turbines is supported by DLR Technology Marketing.

References

- T. Bellmann. Interactive simulations and advanced visualization with modelica. *Proceedings of the 7th Modelica Conference*, pages 541–550, 2009.
- FMI development group. Functional mock-up interface for model exchange and co-simulation. Technical report, MODELISAR consortium, 2014.

- F. Dunne, E. Simley, and L. Pao. Lidar wind speed measurement analysis and feed-forward blade pitch control for load mitigation in wind turbines. Technical report, National Renewable Energy Laboratory, 2011.
- H. Geng, S. Xiao, W. Yang, and G. Yang. Nonlinear dynamic inversion approach applied to pitch control of wind turbines. In *Proceeding of the 11th World Congress on Intelligent Control and Automation*, 2014.
- Martin Hansen. *Aerodynamics of Wind Turbines*. Earthscan, 2008. ISBN 978-1-84407-438-9.
- 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.
- F. Holzapfel. *Nichtlineare adaptive Regelung eines unbemannten Fluggerätes*. Verlag Dr. Hut, 2004. ISBN 9783899631128.
- E. Johnson and A. Calise. Pseudo-control hedging: A new method for adaptive control, 2000.
- J. Jonkman, S. Butterfield, W. Musial, and G. Scott. Definition of a 5-mw reference wind turbine for offshore system development. Technical report, National Renewable Energy Laboratory, 2009.
- H. Joos, J. Bals, G. Looye, K. Schnepfer, 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.
- A. Koerber and R. King. Combined feedback-feedforward control of wind turbines using state-constrained model predictive control. In *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, VOL. 21, NO. 4*, 2013.
- T. Lombaerts, G. Looye, Q. Chu, and J. Mulder. Design and simulation of fault tolerant flight control based on a physical approach. *Aerospace Science and Technology*, 23(1): 151 – 171, 2012. ISSN 1270-9638. 35th ERF: Progress in Rotorcraft Research.
- G. Looye. Design of robust autopilot control laws with nonlinear dynamic inversion. *at Automatisierungstechnik*, 49: 523–531, 2001.
- G. Looye. The new DLR flight dynamics library. *Proceedings of the 6th Modelica Conference*, pages 193–202, 2008.
- 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.
- Modelica Association, editor. *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling Language Specification Version 3.2*. 2010.
- M. Otter, H. Elmqvist, and F. Cellier. Modeling of multi-body systems with the object-oriented modeling language dymola. Technical report, 1996.
- David Schlipf, Tim Fischer, Carlo Carcangiu, Michele Rossetti, and Ervin Bossanyi. Load analysis of look-ahead collective pitch control using lidar. In *Proceedings of the 10th German Wind Energy Conference DEWEK*. Universitaet Stuttgart, 2010.
- Eric Simley, Lucy Y. Pao, Neil Kelley, Bonnie Jonkman, and Rod Frehlich. Lidar wind speed measurements of evolving wind fields. In *Proc. ASME Wind Energy Symposium*, 2012.
- Jean-Jacques E Slotine and Weiping Li. *Applied nonlinear control*. Pearson, Upper Saddle River, NJ, 1991.
- P. Thomas, X. Gu, R. Samlaus, C. Hillmann, and U. Wihlfahrt. The onwind modelica library for wind turbine simulation with flexible structure - modal reduction method in modelica. *Proceedings of the 10th International Modelica Conference*, 2014.
- M. Thümmel, G. Looye, M. Kurze, M. Otter, and J. Bals. Non-linear inverse models for control. *Proceedings of the 4th International Modelica Conference, Hamburg, March 7-8*, 2005.
- J. Tobolar, M. Otter, and T. Bünte. Modelling of vehicle powertrains with the modelica powertrain library. *Systemanalyse in der Kfz-Antriebstechnik IV*, 2007.
- N. Wang and K. Johnson. Lidar-based fx-rls feedforward control for wind turbine load mitigation. In *Proceedings of the 2011 American Control Conference*, 2011.

Free Modelica Library of Chemical and Electrochemical Processes

Chemical 1.1.0

Marek Matejak¹, Martin Tribula¹, Filip Jezek², Jirı Kofranek^{1,2}

¹Institute of Pathological Physiology, 1st Faculty of Medicine, Charles University in Prague
U Nemocnice 5, Prague 2, 128 53, Czech Republic

²Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague,
Technicka 2, Prague 6

marek@matfyz.cz

Abstract

A new, free Modelica library for electrochemical processes has been released - accessible as "Chemical" at <https://www.modelica.org/libraries>. It is based on equilibrating the electrochemical potentials of the substances involved, following the modern theories of physical chemistry. It dynamically solves the chemical equilibration of homogeneous chemical solutions with fully thermodynamic states, supported also through thermal, mechanical, electrical and fluid components of Modelica Standard Library 3.2.1. Even the complex processes can be composed from only a few base components, such as a component for the chemical solution, a component for the chemical substance or a component for the chemical reaction. Behind these components are fundamental laws of thermodynamics and physical chemistry. The library was designed to be very intuitive and easy to use. This paper shows how the library can be used to implement the examples of a lead-acid battery, a hydrogen burning and a chloride shift of human red blood cells.

Keywords: Modelica library, physical chemistry, thermodynamics equilibria, electrochemical potential, electrochemical cell, internal energy, semipermeable membrane

1 Introduction

The content for the chemical library comes from Physiolib (www.physiolibrary.org), a library for physiological calculations (Matejak, 2014; Matejak, et al., 2014). We used Physiolib to implement the most extensive model of human physiology in 2010: HumMod (Hester, et al., 2011; Kofranek, et al., 2011; Matejak and Kofranek, 2011). We named our extended model Physiomod (www.physiomodel.org), and we have continued to extend it at more detailed microscopic and chemical levels. The macroscopic processes and regulations of human physiology are already validated by experiments on animals and humans (Kulhanek, et al., 2010). However, the chemical processes of the models were (until now) conceived in terms of black boxes with inputs and outputs defined more by empirical relation-

ships than by strict physical theory. Focusing on empirical behavior meant that expectations of the elementary processes were well formed. This chemical library allows us to move different substances in different directions across a membrane at the same time, which was not possible when using, for example, the Modelica.Fluid package (Casella, et al., 2006) because stream constructs move all substances together only in the direction of the main solution stream. However having a set of substance connectors (Table 1) there is possible to change each substance separately just by setting its molar flow.

Table 1, Connector for substance: SubstancePort ■ □

nonflow	flow
<i>Electrochemical potential of the substance [J/mol]</i>	<i>Molar flow of the substance [mol/s]</i>

In the Chemical library, we carefully selected only the fundamental definitions from physical chemistry and thermodynamics to derive other known chemical relations (Mortimer, 2008). For example, physical chemistry defines an electrochemical potential $\bar{\mu}_j$ (Eq.1) for each chemical substance j in a homogeneous chemical solution as the composition of a relative molar energy of pure substance μ_j^o (typically tabulated as free molar Gibbs energy of formation), a chemical dissolution component of molar energy $R \cdot T \cdot \ln(a_j)$ (reflecting the mole-fraction based activity of the substance a_j in the solution) and an electrical component of the molar energy $F \cdot z_j \cdot \varphi$ (for substances with charge number z_j in the solution with non-zero electrical potential φ), where T is temperature, R is gas constant and F is Faraday's constant (Eq.1).

$$\bar{\mu}_j = \mu_j^o + R \cdot T \cdot \ln(a_j) + F \cdot z_j \cdot \varphi \quad \text{Eq. 1}$$

The relative energy of the pure substance μ_j^o must be compatible with all tabulated equilibrium coefficients: for example, equilibrium coefficients of chemical reac-

tions (as expressed by the free Gibbs energy of the reaction), Henry's coefficient for gas dissolution equilibrium, Raoult's vapor pressure equilibrium, standard voltages of redox reactions and so on. These known relations do not need to be explicitly written in code because they are the results of algebraic manipulation of the implemented relations, as we mathematically proved during development. Therefore, in this way the Chemical library married chemical, osmotic, thermal, electrical, mechanical and fluid domains. Usage of the library has been very simplified, because it is typically possible to build many types of reactions with few chemical substances - having a set of already defined chemical substances allows automatic calculation of equilibrium coefficients of their chemical processes. The principles that apply to these free Gibbs energies of substances are also applied to free heat energies (free enthalpies) because the same relation — called Hess' law — exists between free enthalpy of chemical processes and relative (free) enthalpies of substances which are typically tabulated as free molar enthalpies of formation. Therefore, the user does not even need to set the value of the heat consumed or released from the chemical process, since this heat energy is automatically derived from the substance definitions.

The development starts with Donnan's equilibria of a semipermeable membrane (Donnan, 1911), together with the Nernst membrane potential, as a consequence of the equilibrated electrochemical potentials of the permeable substances. After these electrochemical processes in a cellular membrane was married with chemical reactions, we realized that the relations are general enough to calculate phase changes, gas solubility, electrochemical cells and other known chemical processes

as described in physical chemistry textbooks, such as (Mortimer, 2008). The result is a library that allows us to create any type of chemical reaction, in any type of homogenous chemical solution. We made it in one hand with thermodynamics and physical chemistry relations behind. In Modelica, the selected base definitions from this theoretical approach can be directly rewritten to the code in their natural mathematical forms, which significantly simplify the implementation.

The Chemical library is freely available at <https://github.com/MarekMatejak/Chemical> and is meant to become a part of Modelica Standard Library.

The library is partially documented directly in the code, more detailed description of the usage, including this article and underlying principles is to be found in attached documents in Documentation folder. This paper shows the main principle and usability of the library on three simple examples.

2 Chemical Substance

The Chemical library in version 1.1.0 contains two basic states of matter: ideal gas and incompressible substance. However, the user can easily (re)define their own state of matter by inserting the correct expressions for the pure substance activity coefficient, molar volume, molar entropy and molar enthalpy, based on the current solution state (temperature, pressure, electric potential and ionic strength) and the substance data. The object-oriented design allows users to define the substance data record as part of the state of matter package, where users can redefine the getter functions of substance properties.

Our examples work with ideal gases in case of all gaseous substance and incompressible state of matter in

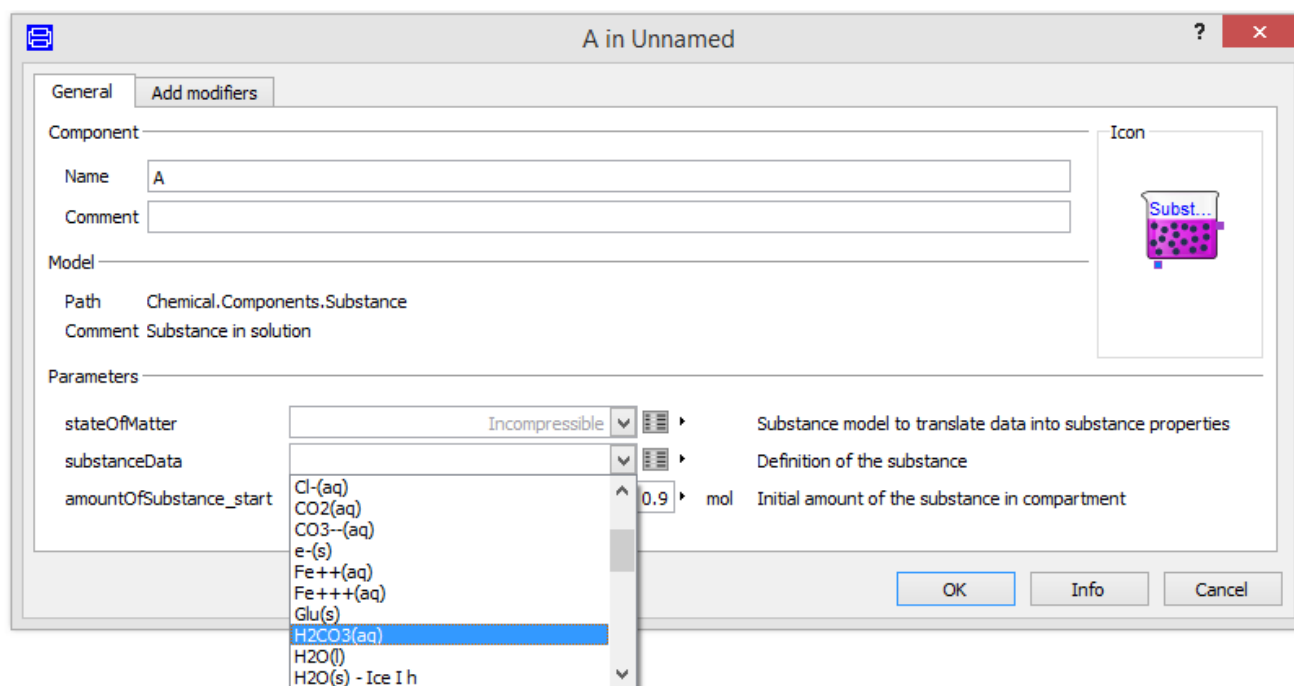


Figure 1. Setting of the predefined chemical substance, where (s) = solid phase, (aq) = dissolved in water, (g) = gas phase and (l) = liquid phase.

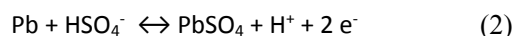
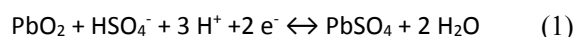
case of liquid or solid. The definition data are the molar mass of the substance, the number of charges of the substance, the molar heat capacity of the substance at a constant pressure, free formation enthalpy, free formation Gibbs energy and density (if incompressible) — all at a temperature of 25°C and pressure 1 bar. Since these parameters are usually recorded in chemical tables at this standard conditions. In this manner, more than 35 real chemical substances in the example package of this chemical library have already been defined. The usage of these predefined substances' data is very simple. In the parameter dialog of the chemical substance, the correct record with this data can be selected, as shown in Figure 1.

This setting is typically the most important setting of each chemical model. All equilibrium coefficients, standard voltages, dissolution coefficients, saturated vapor pressures, etc., are automatically solved using these substance data. As a result, for example, the chemical reaction component only needs to define the stoichiometry coefficients, and the connected substances reach equilibrium.

As a result of fundamental relations, the solution of chemical substances contains enthalpy, entropy and internal energy. These properties can be represented also as Media of MSL 3.2 (e.g. Interfaces.SimpleChemicalMedium). Having solution as homogenous mixture of one state of matter there is an option to use the Fluid connectors and components of MSL 3.2 using Chemical library component named Components.FluidAdapter. The FluidAdapter can connect each substance of the solution with the fluid port, which represent the stream of the whole solution (e.g. Examples.FluidAdapter2).

3 Example of the Lead-Acid Battery

The lead-acid electrochemical cells are characterized by two chemical reactions:



The building of one cell of a lead-acid battery starts with the definition of three solutions: two for the lead electrodes and one for the liquid-acid solution (Figure 2A). This can be done by dragging and dropping the library class 'Components.Solution' into the diagram. We called the first instance "cathode", the second "solution" and the last "anode". We set the parameter 'ElectricalGround' as "false" for all of these solutions in order to attain the possibility of non-zero voltages. Now we can specify the chemical substances inside the chemical solutions. We drag and drop the library class 'Components.Substance' into the "solution" as chemical substances (Figure 2B). H₂O(liquid), H⁺(aqueous) and HSO₄⁻(aqueous) representing the liquid aqueous solution of sulfuric acid. PbSO₄(solid) and PbO₂(solid) are

placed in the "cathode", representing the elements of the positive electrode. The substances Pb(solid) and aPbSO₄(solid) are placed into the "anode", representing the elements of the negative electrode. All of these substances must be given unique names (e.g., "PbSO4" for the cathode and "aPbSO4" for the anode), because the Modelica language does not support two instances with the same name in a single class.

As mentioned above, the appropriate substance data for all these substances must be selected as predefined parametric records, e.g., 'Examples.Substances.Water_liquid', 'Lead_solid', 'Lead_dioxide_solid', 'Lead_sulfate_solid', and so on. The last, very special substance to be included is an electron. This class is called 'Components.ElectronTransfer' and it must be added in order for each electrode to transfer electron from the chemical reaction to the electric circuit (Figure 2C). Each of these substances must be connected to the appropriate solution using a solution port situated in the bottom of the component's icons to indicate that they are all mixed in the solution. By having all these substances, it is possible to implement the chemical reactions. Dragging and dropping the library class 'Components.Reaction' for both chemical reactions, and setting their parameters as an appropriate number of reactants, products and stoichiometry, allows the connection of each substance with the reaction, as expressed in reaction (1) and reaction (2). This setting can be done using the parameter dialog of the cathode chemical reaction (1) as there are four types of substrates (nS=4) with stoichiometric coefficients: one for the first and second reactant, three for the third reactant and two for the fourth reactant (s={1,1,3,2}). There are also two types of products (nP=2) with stoichiometry: one for PbSO₄ and two for water (p={1,2}), following the chemical scheme of the first chemical reaction above. After setting the number of reactants and products, it is possible to connect the substances with reactions. Each instance of reaction has an array of connectors for substrates and an array of connectors for products; the user must be very careful to connect each element of these arrays in the same order as defined by stoichiometric coefficients. This means that, for example, the water must be connected in index 2 to products of the first chemical reaction, because we had already selected the order of products by setting the array of stoichiometric coefficients in reaction (1). The chemical reaction (2) must be set analogically as nS=2, nP=3, p={1,1,2} with connections of substance ports of Pb to substrate[1], HSO₄⁻ to substrate[2], PbSO₄ to product[1], H⁺ to product[2] and e⁻ to product[3], as represented in Figure 2D.

The electrochemical cell has already been implemented at this stage. However, the simulation requires the initial state of substances, which for the fully charged battery means that almost all elements of the cathode are PbO₂ and almost all elements of the anode

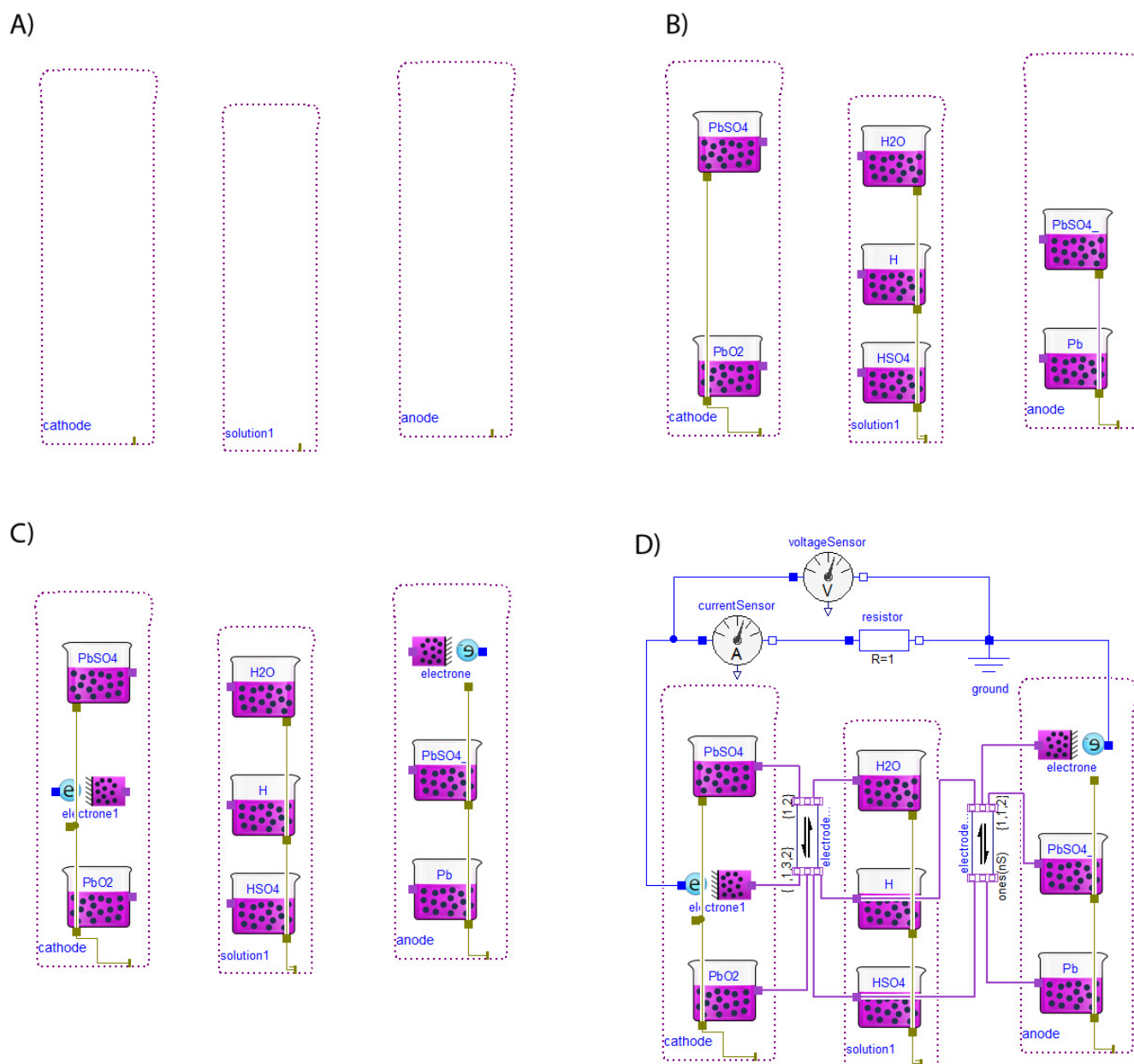


Figure 2. The building of one electro-chemical cell of a lead-acid battery in four steps: A) adding chemical solutions, B) adding chemical substances, C) adding electron transfers and D) adding chemical reactions.

are Pb. In this state, the sulfuric acid can be concentrated, which increases the effectiveness of the electro-chemical cell. To set this state, it is possible to just double-click on PbO₂ and Pb and set the amount, e.g., 1mol. To set the pure concentrated sulfuric acid we can also set the amount of SO₄⁻ and H⁺ as 1mol. This fully charged ideal state is ready to simulate when it is connected to the electric ground via one of the electric ports of the one electron transfer component.

These batteries can be connected to any electrical circuit that is slowly discharging. For example, if we only connect the simple electric resistance of 1 Ohm as expressed in Figure 2D, then the simulation of the discharging process over 13 hours and 45 minutes gives the results of electric current and electric potential, as can

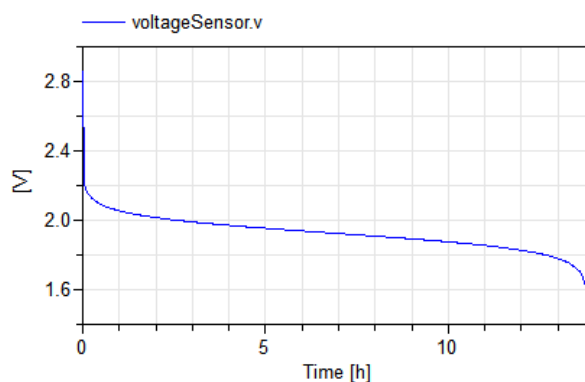
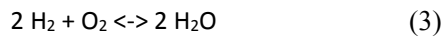


Figure 3. Discharging simulation of the lead-acid battery cell from Figure 2D, with the initial amount of substances as described in the text.

be seen in Figure 3. The exchange of the resistor with a voltage source can simulate the charging process for a discharged cell.

4 Example of the Hydrogen Burning

In contrast with oxidation-reduction reactions, describing processes in lead-acid electrochemical cells, the gaseous reaction of burning hydrogen is very simple:



However, this reaction generates a large amount of energy which can be used for mechanical or thermal purposes.

Building this model (Figure 4) using the Chemical library is very easy. First, we drag and drop the library class ‘Components.Solution’ into the diagram of our new model, labeled ‘idealGas’ in Figure 4. In parameter dialog of this solution we check “useThermalPorts” and “useMechanicsPorts” to enable the thermal and mechanical interface. In the same dialog we need to set the area of the piston (e.g., 1 dm²), where the pressure provides the force of the green mechanical port of the uppermost side. The next parameter is the ambient external pressure surrounding the system (e.g., 1 bar). All three chemical substances of the reaction (1) can be added by

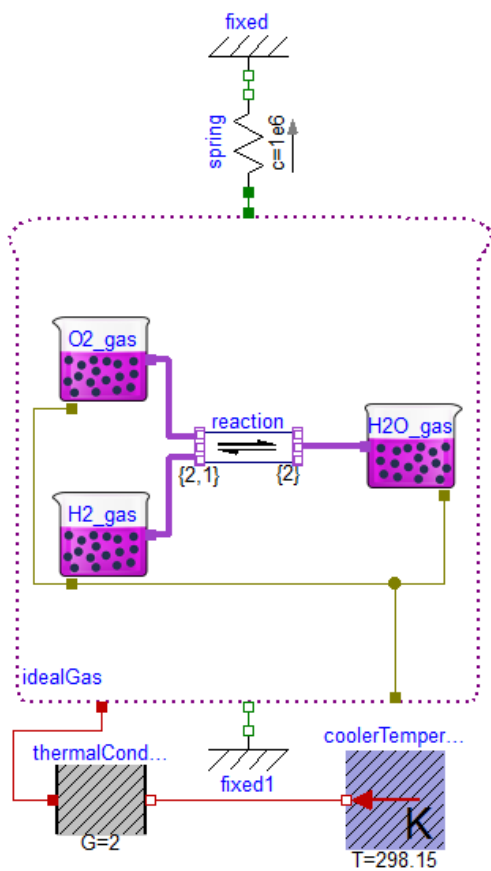


Figure 4. A hydrogen-burning piston with the spring above the piston and cooling to provide an environment with a constant temperature.

dragging and dropping the library class ‘Components.Substance’. Because this model uses gases, the state of matter must be changed to some gas, such as the ideal gas prepared as ‘Interfaces.IdealGas’. The substance data must be selected to define the appropriate substances such as ‘Hydrogen_gas’, ‘Oxygen_gas’ and ‘Water_gas’ in package ‘Examples.Substances’. In addition, the initial amounts of substances can be prepared for the ideal solution of hydrogen and oxygen gases at a ratio 2:1 to attain the chemical equation above, with the expectation that at the end of the burning process, only water vapor would be presented. Therefore, the initial values of H_2 particles could be set to 26 mmol and of O_2 particles as 13 mmol. All substances must be connected with the ‘idealGas’ using the blue colored solution port situated on the bottom side of each substance and solution. Then, the chemical reaction is inserted into the diagram of this model as library class ‘Components.Reaction’, and it is set to two substrates ($nS=2$) with stoichiometry $s=\{2,1\}$ and one product with stoichiometry $p=\{2\}$ to represent the reaction (3). The substances are then connected using violet colored substance connectors with appropriate indexes: H_2 to substrates[1], O_2 to substrates[2] and H_2O to products[1]. At this point, the

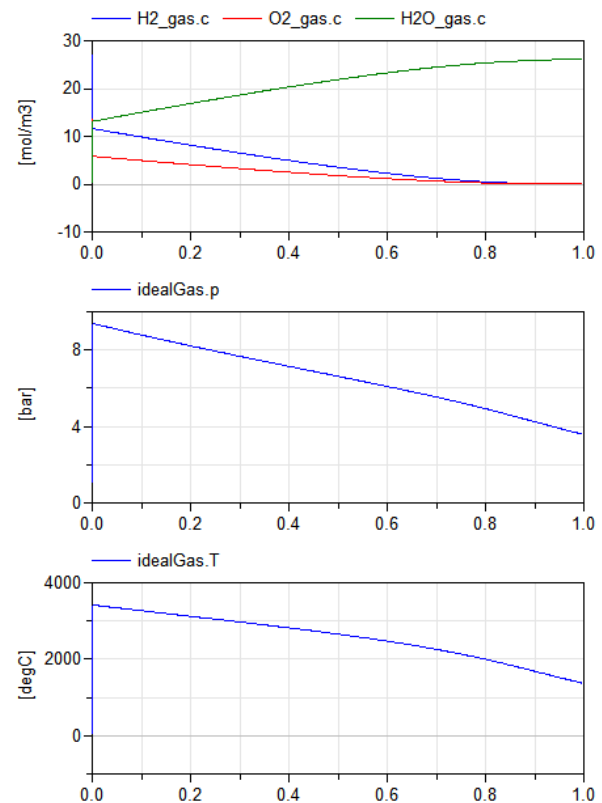


Figure 5. Simulation of the hydrogen-burning experiment in Figure 4. The initial phase of the explosion occurs very rapidly — the temperature reaches immediately 3600°C from 25°C and the pressure reaches 10 bars from 1 bar. This pressure and this temperature are generated because of a very strong spring, which allows the volume to change only by about 8% during the explosion.

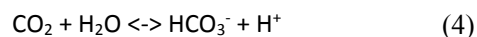
model is prepared to simulate the conditions of an unconnected heat port and an unconnected mechanical port. This simulation reaches the theoretical ideal of thermally isolated (zero heat flow from/to the solution) and isobaric (zero force generated on piston) conditions.

However, in the real world, there is always some thermal energy flow from the solution, and this cooling process can be connected using the thermal connector of the Modelica Standard Library 3.2.1. For example, the simple thermal conductor of thermal conductance 2W/K at a constant temperature environment of 25°C is represented in Figure 4. The mechanical power of the engine can be connected to the robust mechanical model. However, in our example we selected only a very strong mechanical spring with a spring constant of 10^6 N/m to stop the motion of the piston in order to generate the pressure. This standard spring component is situated above the solution in Figure 4. The results of this experiment are shown in Figure 5.

5 Example of Chloride Shift

The mature red blood cell (erythrocyte) is the simplest cell in the human body. Its primary function is the transportation of blood gases, such as oxygen O_2 (from the lungs to tissues) and carbon dioxide CO_2 (from tissues to the lungs). The chemical processes behind the gases'

transportation are complex because the capacity of water to transport their freely dissolved forms is very low. To transport sufficient amounts of O_2 and CO_2 , the gases must be chemically bound to hemoglobin such as described in (Matejak, et al., 2015) and/or transported as different substances, which can be present in water in much higher concentrations than their freely dissolved forms allow. Therefore, to transport a sufficient amount of CO_2 , it must be changed to HCO_3^- using the chemical reaction:



This reaction takes place mainly inside the red blood cell, because only here it is presented with the enzyme carbonic anhydrase. Therefore, the increase of total carbon dioxide content of blood in tissues and its decrease in lungs are always connected with the chloride shift between blood plasma and the intracellular fluid of erythrocytes, as represented in Figure 6.

The blood plasma and intracellular fluid are divided by the cellular membrane composed of a special, very compact lipid double-layer. A lipophobic compound (not soluble in lipids) cannot cross the membrane without special proteins called membrane channels. Even water molecules must have membrane channels (called aquaporins) in order to cross the cellular membrane. In

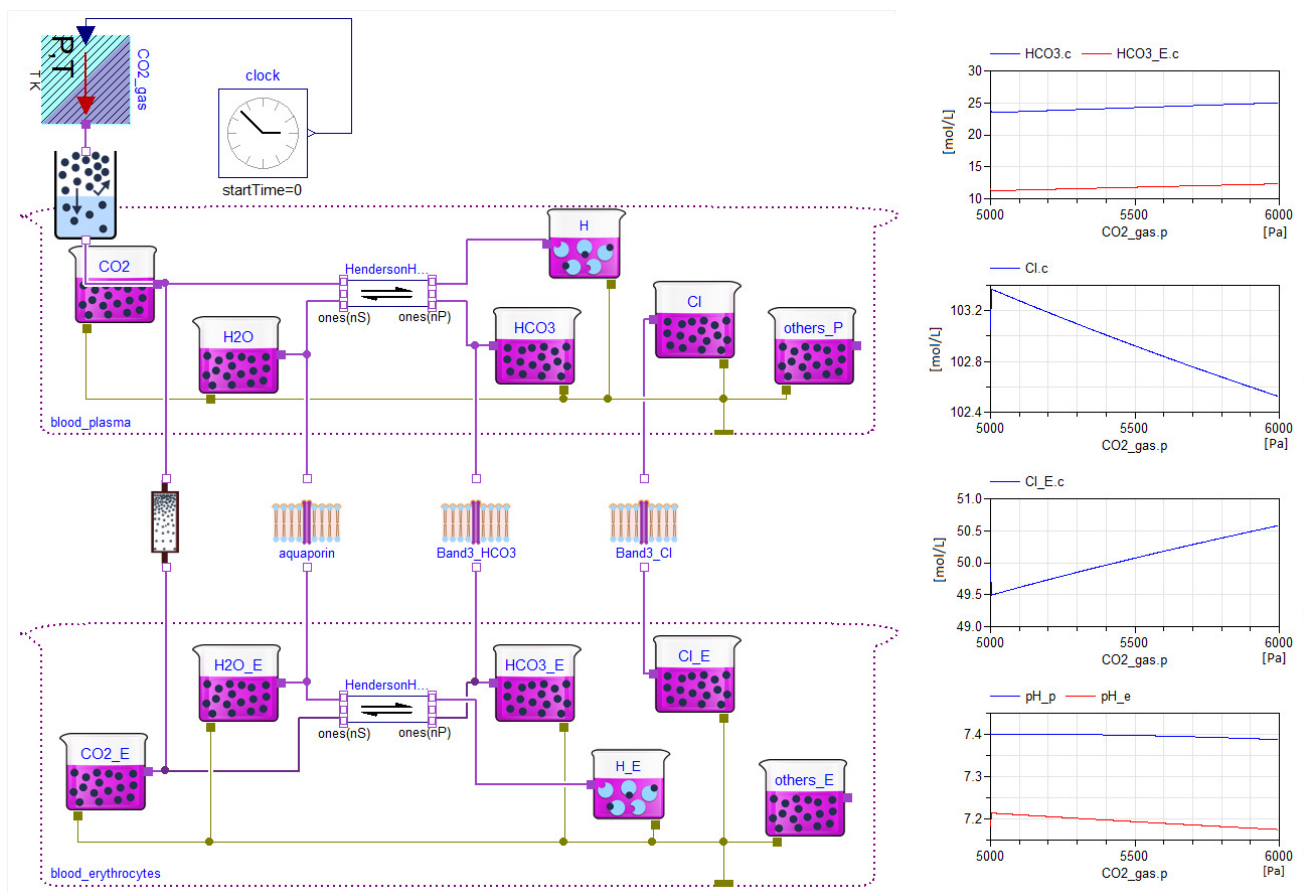


Figure 6. Chloride shift with carbon dioxide hydration with assumption of non-bicarbonate linear acid-base buffering properties of plasma and erythrocytes.

addition, the chloride shift (also known as the Hamburger shift) is exchanging an aqueous chloride Cl^- for an aqueous bicarbonate HCO_3^- in both directions across the cellular membranes of red blood cells using the membrane channel “Band 3”. Each passive membrane channel only allows the equilibration of the electrochemical potentials of the specific permeable ions on both sides of membrane. The different electric potentials on each side of membrane allow their different concentrations to achieve equilibrium.

Conversely, the solution’s equilibrium of different ions’ compositions on both sides of the membrane creates the measurable electric membrane potential. This process is not so intuitive, because even though neither solution needs to have an electric charge, there can be a non-zero electric potential for permeable ions. This potential for permeable ions at equilibrium is called the Nernst membrane potential and, in the Chemical library, it is a direct mathematical result of the equality of the electrochemical potential of the ion in both solutions.

The intracellular solution must be set at the possible nonzero electric potential (`ElectricalGround=false`) because, as a result, the membrane potential of the erythrocytes is calculated as -12mV , which agrees with experimental data by Gedde and Huestis (Gedde and Huestis, 1997) in the electrolytes’ setting by Raftos et al. (Raftos, et al., 1990).

In this way, it is possible to model more complex processes of a membrane where chemical reactions of active membrane channels or membrane receptors can both be used.

6 Discussion

Nowadays, alternative free Modelica libraries for chemical calculations exist, such as `FCSys v0.2`, `FuelCellLib 1.0`, `Modelica_EnergyStorage v3.2.1`, `BioChem v1.2` or our `Physiolibrary v2.3`. However, we are not satisfied with these libraries, because none of them are based on equilibrating electrochemical potentials. This lack makes it difficult to establish real equilibria in electrochemical processes, and we believe that it is very difficult to implement any kinetics without realistic equilibria.

This new chemical library is more suited to understanding the detailed electrochemical environment of human cells and cellular electrochemical processes, a task at which the `Physiolibrary` failed. For example, we found that the equilibrium of osmolarities (as validated and verified for macroscopic and capillary membranes) was not in good agreement with measured data of cellular membranes. The real data of human blood include the total molarity of plasma at 289 mmol/L and the molarity of intracellular space of erythrocytes at 207 mmol/L at osmotic equilibrium, as presented by Raftos et al. (Raftos, et al., 1990). These values are definitely not the same, and the explanation for these disproportions can be found in physical chemistry (Mortimer,

2008). However, when the electrochemical potential from the original data was calculated, it was found that electrochemical potential is in equilibrium instead of a state of osmolarity. Therefore, equilibrating the electrochemical potential instead of osmolarity can help us to describe each type of membrane and each type of substance, reaching the expected values as measured in osmotic experiments for both organ and cellular membranes.

The library is usable for any chemical or electrochemical process. However, chemical kinetics are not yet seriously validated, so the only assumption is, that the equilibrating time of chemical processes is by orders of magnitude shorter than of other connected domains. Testing has been done through examples in examples package in Dymola 2015.

The mentioned examples, together with many others that have been processed, are implemented and tested in the ‘Example’ package of the library. They are the definition of a very simple and general chemical reaction and also the complex models, such as: the heating of water solutions, an exothermic reaction, the vaporization of water, O_2 and CO_2 gas solubility in aqueous solutions, an enzymatic reaction, a Harned cell (such as the typical pH measurement of an electrochemical cell), water self-ionization, carbon dioxide in a water solution, inorganic phosphate in a water solution, the albumin (blood plasma protein with 218 sides for the binding of H^+) titration model by Figge-Fencl and allosteric models of hemoglobin oxygenation by Monod-Wyman-Changeux. All of these examples illustrate usage of the chemical library’s components, such as the chemical solution, chemical substance and chemical reaction.

We hope, that with reference to the tabulated thermodynamic properties of organic substances, it should be also possible to implement even a complex metabolic, regulations and neural pathways of human physiology using this Chemical library.

7 Acknowledgements

The authors appreciate the partial funding of this work by PRVOUK P/24/LF1, SVV 260157/2015 and FR Cesnet 551/2014.

References

- Casella, F., *et al.* The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. In, *Proceedings of the Modelica Conference*. 2006. p. 631-640.
- Donnan, F.G. Theorie der Membrangleichgewichte und Membranpotentiale bei Vorhandensein von nicht dialysierenden Elektrolyten. Ein Beitrag zur physikalisch-chemischen Physiologie. *Zeitschrift für Elektrochemie und angewandte physikalische Chemie* 1911;17(14):572-581.

- Gedde, M.M. and Huestis, W.H. Membrane potential and human erythrocyte shape. *Biophys. J.* 1997;72(3):1220.
- Hester, R.L., *et al.* HumMod: a modeling environment for the simulation of integrative human physiology. *Front. Physiol.* 2011;2.
- Kofránek, J., Mateják, M. and Privitzer, P. HumMod - large scale physiological model in Modelica. In, *8th International Modelica Conference*. Dresden, Germany; 2011.
- Kulhánek, T., *et al.* Distributed computation and parameter estimation in identification of physiological systems. In, *VPH conference*. 2010.
- Mateják, M. Physiology in Modelica. *Mefanet J* 2014;2(1):10-14.
- Mateják, M. and Kofránek, J. HumMod–Golem Edition–Rozsáhlý model fyziologických systémů. *Medsoft* 2011:182-196.
- Mateják, M., Kulhánek, T. and Matoušek, S. Adair-based hemoglobin equilibrium with oxygen, carbon dioxide and hydrogen ion activity. *Scand. J. Clin. Lab. Invest* 2015;75(2):113-120.
- Mateják, M., *et al.* Physiobrary - Modelica library for Physiology. In, *10th International Modelica Conference*. Lund, Sweden; 2014.
- Mortimer, R.G. Physical Chemistry (Third Edition). In: Mortimer, R.G., editor. Burlington: Academic Press; 2008. p. 1-1385.
- Raftos, J.E., Bulliman, B.T. and Kuchel, P.W. Evaluation of an electrochemical model of erythrocyte pH buffering using ³¹P nuclear magnetic resonance data. *The Journal of general physiology* 1990;95(6):1183-1204.

Modeling Biology in Modelica: The Human Baroreflex

Christopher Schölzel¹ Alexander Goesmann² Gernot Ernst³ Andreas Dominik¹

¹KITE, Technische Hochschule Mittelhessen, Giessen, Germany,
{christopher.schoelzel, andreas.dominik}@mni.thm.de

²Justus Liebig University Giessen, Giessen, Germany

³Vestre Viken Hospital Trust, Kongsberg, Norway

Abstract

Systems biology is a field that requires complex multi-scale models of systems that are evolved rather than engineered. No unifying theory exists for biology as it does for engineering domains. Thus, models appear in very diverse forms. Components can be genes, cells, organs or even whole ecosystems. These components can intuitively be represented as classes in an object-oriented language, making systems biology a perfect application for Modelica. However, we still only see very few models from this domain. In an attempt to change this, we show that Modelica can exactly reproduce the simulation results of a reference implementation of an established biological model of the human baroreflex. Our implementation highlights the strengths of Modelica like the event finding mechanism, which makes the model more precise. We also show that biological systems pose interesting challenges like signals with non-uniform delays and the interaction of complex rhythms.

Keywords: systems biology, baroreflex, cardiovascular system, heart rate variability

1 Introduction

Biological systems are complex, dynamic and packed with feedback loops. Even small academic examples of systems that exhibit these properties are very hard to understand and analyze for humans without proper tools (Voit, 2013, pp. 8–10). Recent support comes in form of mathematical models, forming the field of systems biology. A strong mathematical foundation can help where intuition fails and indeed there are now projects such as the Virtual Liver Network (Holzhütter et al., 2012), the Blue Brain Project (Markram, 2006) or the Physiome Project (Hunter et al., 2002) that are on the way of building comprehensive multi-scale models for complete human organs.

In all of these projects, communication between models at different scales of time and space is a key challenge. Low-level models of biochemical reactions have to be integrated into models on the cellular level which

then again need to be composed together to reach the desired level of abstraction. This is made more difficult by the fact that there exists no unifying theory in biology as it does in other domains such as electrical engineering (Voit, 2013, pp. 413–415). It is often not possible to build biological models as a bottom-up approach from the biochemical or genetic level, because too much is still unknown. For example, even when we narrow down the problem to these two lowest levels we still only begin to understand the role of long noncoding RNAs (Ponting et al., 2009) or glycoproteins (Ranzinger and York, 2012) in the regulation of cellular processes. Instead, researchers such as Denis Noble suggest a “middle-out” approach, that starts at the layer of abstraction where most experimental data is available (Noble, 2002).

There exist projects that aim to provide a common language for biological modeling, the most prominent being the Systems Biology Markup Language (SBML). SBML is an XML-based description format for models featuring the biochemical concept of different substances (called *species*) and *reactions* that modify the quantity of these substances. Support for the SBML is widespread in the systems biology community, but two main factors limit its usefulness for multi-scale modeling and the middle-out approach. Firstly, SBML is a data format and not a programming language. Mathematical formulas have to be specified in MathML which is not a human-readable format. Therefore, often multiple tools are needed for the generation and simulation of SBML models. Secondly, SBML is specifically designed for modeling biochemical processes which enforces a bottom-up approach and makes it impossible to start the modeling process at a higher level of abstraction. Other languages that are currently used in systems biology include numerical computing environments like Matlab and Mathematica, and general purpose programming languages like C. These languages all have sufficient expressive power and flexibility to start the modeling process at any layer of abstraction, but they also come with a lot of cognitive overhead. Scientists interested in modeling biological systems, such as physicians or biologists, have to build their equation systems and solve them by hand, or fit them

into a specific structure for existing implementations of the desired solver. As a result, the structure of the model is fitted to the programming platform instead of the other way around. It also makes the model harder to understand and discuss both for language experts that are not familiar with the modeled system and biologists that are not familiar with the specific language constructs. Especially with large equation systems, one has to put a lot of effort into structuring his code to preserve a clear distinction between different components of the modeled systems, let alone different abstraction layers of one and the same component.

Modelica offers an elegant solution for these problems. As an object-oriented language it makes the encapsulation of subsystems into larger components intuitive and thus highly facilitates multi-scale modeling. At the highest level, a Modelica model may present cells or entire organs as components with a clear interface that hides the details of the implementation at lower levels. If desired, however, a user of the model always has the possibility to inspect a component and dig one level deeper to look at the subsystems constituting the organ or cell. In theory, this makes it possible to build arbitrarily deep nested structures without overwhelming the model user with too much detail at a single level. Additionally, the declarative acausal nature of Modelica allows to write most formulas verbatim in the same way as they may appear in the mathematical definition of the model. This greatly reduces the cognitive overhead necessary to understand, maintain and extend an existing model.

Some projects already use Modelica for tasks related to systems biology. The BioChem library allows to translate SBML models to Modelica and vice versa offering a starting point for system biologists interested in Modelica (Nilsson and Fritzson, 2005). Additionally the PhysiLibrary by Matejak et al. (2014) and the HumanLib by Brunberg and Abel (2010) are both targeting the modeling of the human physiology. Yet, when we looked at three recent systems biology textbooks, we did not find any reference to Modelica (Voit, 2013; Klipp et al., 2011; Kremling, 2012), although one of these books featured a list of over 80 tools for modeling in systems biology, including Matlab, Mathematica, SBML and a variety of application-specific SBML tools (Klipp et al., 2011).

In our opinion not only system biologists can benefit from Modelica, but also the field of systems biology provides interesting challenges for Modelica modelers. In contrast to most other application areas of Modelica, biological systems are evolved rather than engineered. Specifically, this means that there is usually a high level of complicated communication between multiple parts of the system and that these interactions are not always straightforward. One major example is the the “central dogma” of molecular biology. It states that every aspect of a living system can be explained starting from the DNA which is translated to proteins. These proteins then carry out some function in the system, but do not change

the genetic code. This is a natural assumption that would seem intuitive to an engineer or computer scientist: A source code defines programs regulating the behavior of a system. However, the study of *epigenetics* shows that there are many factors that can influence gene expression and thus change the way in which the DNA-code is read (Holliday, 2006). To make things more difficult, there is no moment in the life of an organism where a cell is constructed from nothing but DNA. Even a single fertilised egg cell still has not only inherited the DNA from its parents but also all of the other biochemical substances in this cell.

Therefore, when we build models of biological systems, we might encounter unusual connection patterns between components. These new types of problems may indicate areas, where Modelica still can be improved. As an additional argument for the study of biological models, the field of systems biology spans a large area of interesting and relevant topics, from the modeling of brain activity to finding diagnosis criteria and treatments for cardiac diseases, diabetes or cancer to the modeling of whole ecosystems like oceans (Voit, 2013, pp. 399–415). The potential of applying mathematical modeling to biological systems is vast, and with Modelica we can facilitate the generation of new insights.

With this paper we want to take a further step towards bringing together Modelica modelers and systems biologists. We show that it is not only possible to convert SBML models to Modelica with the BioChem library or build physiological models from predefined components with the PhysiLibrary, but also to implement a biological model in Modelica directly from the mathematical description. As a proof of concept, we therefore implemented an established model of the human baroreflex by Seidel (1997) in Modelica and compared it with the original implementation of Seidel written in C. An introduction to the Seidel-Herzel model (SHM) will be given in section 2. The two implementations of the SHM – hereinafter called SHM-M for our Modelica version and SHM-C for Seidels C implementation – will both be described in section 3. To demonstrate that Modelica is indeed flexible enough to precisely reflect the mathematical description of the model, we directly compare the output of SHM-C and SHM-M in section 4 followed by a discussion of the results in section 5. There we also demonstrate that biological models like the SHM fit nicely into the modeling paradigms of Modelica and highlight some interesting challenges of the implementation process. Finally, a short conclusion can be found in section 6.

2 The Seidel-Herzel model

The Seidel-Herzel model (SHM) is a model of the human baroreflex that was created by Henrik Seidel and Hanspeter Herzel and first published in 1995 with the

main purpose of analyzing heart rate variability (HRV) (Ernst, 2014). There are two main reasons why we chose this model as example for a typical biological system that can be implemented in Modelica: Firstly, although the cardiovascular system is well researched, heart diseases are still the most common cause of death worldwide (Naghavi et al., 2014). This means that models like the SHM are still relevant and may even help towards finding diagnostic criteria for heart-related diseases. The second reason to choose the SHM over other models of the human heart was that it is rather compact but still covers the effects of multiple organs. The doctorate thesis of Seidel contains the most recent version of the model, which has 24 equations and 52 parameters (Seidel, 1997).

Although there is another publication by Seidel *et al.* from 1998 (Seidel and Herzog, 1998), we used this version because it features several improvements of the model that were not present in the journal publication. In the following, we will give a short introduction into the components of the SHM and also explain its relevance in literature. For a more detailed explanation of the formulas, the reader is referred to Seidel's doctorate thesis (Seidel, 1997). This version of the SHM considers the physiological effects of the baroreceptors in the blood vessels, the autonomic nervous system, the lung, the sinus- and av-nodes, the heart itself and the Windkessel arteries. It does not introduce different compartments in the blood system but instead models the arterial blood pressure as a single physical quantity.

2.1 Baroreceptors

Baroreceptors are the sensory neurons measuring the pressure in a blood vessel. The basic neural firing frequency of the baroreceptors v_b in the SHM is calculated with the following formula.

$$v_b'(t) = p - p_b^{(0)} + k_b^{dp} \frac{dp}{dt} \quad (1)$$

This includes the effects that baroreceptors respond to the static blood pressure level p as well as to an increase or decrease in blood pressure and that they only respond to blood pressure levels above a threshold $p_b^{(0)}$. The parameter k_b^{dp} is a scaling factor to adjust the relative influence of the blood pressure slope.

To account for the saturation effect of baroreceptors, this value is passed through the saturation function

$$v_b(t) = p_b^{c0} \left(1 + \tanh \left(\frac{v_b'(t) - p_b^{c0}}{p_b^{c0}} \right) \right) \quad (2)$$

$$p_b^{c0} = p_b^c - p_b^{(0)} \quad (3)$$

where p_b^c is a scaling parameter to adjust the maximum of the saturation function, which lies at $2p_b^{(0)}$.

In a living organism, however, the signal of baroreceptors at different parts of the body reach the autonomic nervous system (ANS) at different time instants. This effect is modeled in the SHM with a broadening function that is additionally applied to the saturated baroreceptor response.

$$\tilde{v}_b(t) = \int_{-\infty}^{\infty} g(t - \tau) v_b(\tau) d\tau \quad (4)$$

$$g(t) = \begin{cases} 0 & \text{for } t \leq 0 \\ \frac{1}{\sigma} \chi_{2+\frac{\eta}{\sigma}}^2 \left(\frac{t}{\sigma} \right) & \text{for } t > 0 \end{cases} \quad (5)$$

In this equation χ_n^2 is the probability distribution function of the chi squared distribution and σ and η are scaling parameters to adjust the broadening range.

2.2 Lung

The Lung influences the heart rate both through neural signals and the mechanical pressure in the thorax. The SHM assumes a constant breathing rate that is only modified by a noise term. The activity of respiratory neurons $v_r(t)$ is given by

$$v_r(t) = \frac{1}{2} (1 - \sin(2\pi\phi_r(t))) \quad (6)$$

$$\phi_r(t) = \frac{t - (t_{r,i} - \theta_r)}{T_{r,i}} \quad (7)$$

where $t_{r,i}$ is the beginning of the last inspiration phase and θ_r is a phase shift parameter that determines the time between the firing of respiratory neurons and the actual mechanical movement of the lungs.

The mechanical respiratory influence $f_m(t)$ is defined similarly by the following equation.

$$f_m(t) = -\sin(2\pi\phi_r(t - \theta_r)) \quad (8)$$

Even during voluntarily controlled breathing, the breathing rate of a human is always subject to fluctuations. Seidel models these fluctuations by introducing a noise term which is applied to the mean breathing rate \bar{T}_r at each breathing cycle with an autoregressive function.

$$T_{r,i} = k_{T_r} \bar{T}_r + k_{T_r}^{\text{last},1} T_{r,i-1} + k_{T_r}^{\text{last},2} T_{r,i-2} + \sigma_{T_r} \xi \quad (9)$$

$$k_{T_r} = (1 - k_{T_r}^{\text{last},1} - k_{T_r}^{\text{last},2}) \quad (10)$$

In this formula, $T_{r,i}$ is the breathing period at the i th breathing cycle; $k_{T_r}^{\text{last},1}$ and $k_{T_r}^{\text{last},2}$ are parameters that determine the influence of the last and second last breathing period on the current period; and σ_{T_r} is the amplitude of the white noise ξ .

2.3 Autonomic Nervous System

The autonomic nervous system (ANS) consists of the sympathetic and the parasympathetic system. The sympathetic system increases the heartbeat frequency through the release of norepinephrine as neurotransmitter (via synapses) and as Hormone (via the blood vessels). The parasympathetic system has inhibitory influence on the heartbeat frequency through the release of the neurotransmitter acetylcholine.

The formulas for the neural activity of the sympathetic system v_s and the parasympathetic system v_p therefore only differ by the sign with which the baroreceptor activity enters the equation.

$$v'_s(t) = v_s^{(0)} - \left(1 + k_s^{br} v_r(t)\right) \tilde{v}_b(t) + k_s^r v_r(t) \quad (11)$$

$$v'_p(t) = v_p^{(0)} + \left(1 + k_p^{br} v_r(t)\right) \tilde{v}_b(t) + k_p^r v_r(t) \quad (12)$$

$$v_s(t) = \max(0, v'_s(t)) \quad (13)$$

$$v_p(t) = \max(0, v'_p(t)) \quad (14)$$

Both equations have a base firing rate $v_{s/p}^{(0)}$ and scaling parameters $k_{s/p}^r$ for the respiratory influence and $k_{s/p}^{br}$ for the correlation between the activity of the baroreceptors and the respiratory neurons.

2.4 Substance Concentrations

The SHM models several concentrations of neurotransmitters and hormones. The concentration of norepinephrine at the sinus node (c_{sNe}) directly influences the pacemaker phase together with the concentration of acetylcholine (c_{sAc}) at the sinus node. Additionally, norepinephrine can also act as a hormone. The ventricular concentration c_{vNe} in the heart itself increases the contractility (force of the contraction). The concentration in the Windkessel arteries c_{wNe} increases the stiffness of the vessel walls, resulting in a higher blood pressure during the diastole.

The release of this concentration is triggered by one neural signal and can be inhibited by another neural signal. For norepinephrine the excitatory signal comes from the sympathetic system while the parasympathetic system inhibits the release. For acetylcholine the parasympathetic system is the excitatory part and there is no inhibition modeled. Both inhibitory and excitatory signals only take effect after a delay θ_c^x and are subject to saturation. This leads us to the following equations

$$\text{ex}_c^x(t) = \text{in}_c^x(t) = \tanh(k_c^x v_x(t - \theta_c^x)) \quad (15)$$

$$\tau_{sNe} \frac{dc_{sNe}}{dt} = -c_{sNe}(t) + \text{ex}_{sNe}^s(t) (1 - \text{in}_{sNe}^p(t)) \quad (16)$$

$$\tau_{vNe} \frac{dc_{vNe}}{dt} = -c_{vNe}(t) + \text{ex}_{vNe}^s(t) (1 - \text{in}_{vNe}^p(t)) \quad (17)$$

$$\tau_{wNe} \frac{dc_{wNe}}{dt} = -c_{wNe}(t) + \text{ex}_{wNe}^s(t) (1 - \text{in}_{wNe}^p(t)) \quad (18)$$

$$\tau_{sAc} \frac{dc_{sAc}}{dt} = -c_{sAc}(t) + \text{ex}_{sAc}^p(t) \quad (19)$$

where the τ_c and the k_c^x are scaling parameters for the overall slope of the concentrations $c_c(t)$ and the influence of the inhibitory or excitatory signal.

2.5 Sinus Node

The sinus node is the main pacemaker of the heart. In the SHM it is modeled with the pacemaker phase $\phi(t)$ which generates a sinus signal when its value becomes one and is then directly reset to zero. The rate of the pacemaker phase increases with an increased concentration of norepinephrine and decreases with an increase in acetylcholine. The latter is additionally modified by a ‘‘phase-effectiveness curve’’ $F(\phi)$, because the effect of the parasympathetic signal on the pacemaker changes with the phase of the heart cycle. The resulting behavior is given by the following formula

$$\frac{d\phi}{dt} = \frac{1}{T^{(0)}} \left(1 + k_\phi^{sNe} c_{sNe}(t) - k_\phi^{sAc} c_{sAc}(t) \frac{F(\phi(t))}{\bar{F}_\phi} \right) \quad (20)$$

$$\bar{F}_\phi = \int_0^1 F(\phi) d\phi \quad (21)$$

$$F(\phi) = \phi^{1.3} (\phi - 0.45) \frac{(1 - \phi)^3}{(1 - 0.8)^3 + (1 - \phi)^3} \quad (22)$$

where $T^{(0)}$ is the duration of the heart cycle without any input from the ANS and k_ϕ^{sNe} and k_ϕ^{sAc} are scaling parameters for the influence of the concentrations of norepinephrine and acetylcholine.

The heart period of a human is always subject to additional fluctuations that do not originate from breathing or the signals of the ANS. These influences can be implemented by replacing the parameter $T^{(0)}$ with a noisy base period $T_n^{(0)}$, which varies with the heartbeat number n similarly to the respiratory base period in Equation 9.

$$T_n^{(0)} = \bar{T}^{(0)} + k_{T^{(0)}}^{\text{last}} (T_{n-1}^{(0)} - \bar{T}^{(0)}) + \sigma_{T^{(0)}} \xi \quad (23)$$

2.6 Contraction Model

Not every sinus signal generates a heartbeat and not every heartbeat is triggered by a sinus signal. On the one hand, if there is no sinus signal for a prolonged time period, the atrioventricular node (AV node) will trigger a

contraction by itself. This is represented by the parameter T_{av} so that a contraction is triggered if more than T_{av} seconds have passed since the last contraction at time $t_{c,n}$. On the other hand, a sinus signal does not immediately correspond to the beginning of a contraction. There is an atrioventricular conduction delay $T_{avc,n}$ that passes from the firing of the sinus node (which is located at the atrium) at time $t_{s,n}$ to the contraction of the ventricles. It depends on the time that has passed since the last contraction at $t_{c,n}$. Additionally, the sinus node has a refractory period T_{refrac} during which no new signal may be generated. Combining these two effects, we receive the following equation for atrioventricular conduction time.

$$T_{avc,n} = \begin{cases} T_{avc}^{(0)} + k_{avc}^t e^{-\frac{(t_{s,n}-t_{c,n})}{\tau_{avc}}} & \text{if } t_{s,n} - t_{c,n} > T_{refrac} \\ \infty & \text{else} \end{cases} \quad (24)$$

In this equation $T_{avc}^{(0)}$ is the base value for the atrioventricular conduction time, k_{avc}^t is a scaling parameter for the influence of the time that has passed since the last contraction and τ_{avc} is a reference value for the atrioventricular conduction time.

2.7 Heart

The final components of the SHM are the contraction of the heart and the Windkessel arteries that are responsible for the blood pressure increasing during the systole and decreasing during the diastole. The switch between systole and diastole is modeled explicitly by a fixed systole duration of τ_{sys} . During the systole from $t_{c,n}$ to $t_{c,n} + \tau_{sys}$ the blood pressure follows the equations

$$\frac{dp}{dt} = \frac{1}{\tau_{sys}} \frac{S_n}{C} (1 - \phi_{sys}(t)) e^{1-\phi_{sys}(t)} \quad (25)$$

$$\phi_{sys}(t) = \frac{t - t_{c,n}}{\tau_{sys}} \quad (26)$$

where C is a scaling constant for the contractility S_n . The value of S_n is determined at the beginning of the systole at $t_{c,n}$ as follows.

$$S_n = S^{(0)} + (k_S^{vNe} c_{vNe}(t_{c,n}) + k_S^m f_m(t_{c,n})) S^{(T_{n-1})} \quad (27)$$

$$S^{(T_{n-1})} = \left(1 - \left(1 - \min \left(1, \frac{t_{c,n} - t_{c,n-1}}{\hat{T}} \right) \right)^2 \right) \quad (28)$$

During the diastole, the equation for the blood pressure switches to the following formula that accounts for the effect of the Windkessel arteries. These arteries directly connected to the heart are elastic and act as a

dampening system. During the systole they “store” blood by expanding the blood vessels. During the diastole they contract back to their original state slowly releasing the stored blood.

$$\frac{dp}{dt} = \frac{-(p - p_w^{(0)})}{\tau_w(t)} \quad (29)$$

$$\tau_w(t) = \tau_w^{(0)} + k_w^{wNe} c_{wNe}(t) \quad (30)$$

In this formula $p_w^{(0)}$ is the minimum blood pressure that is still present even if the Windkessel arteries are fully relaxed and the heart does not pump, $\tau_w^{(0)}$ is a base value for the time needed for the Windkessel arteries to fully relax, and k_w^{wNe} is a scaling factor for the influence of the norepinephrine concentration in the arteries on this relaxation time.

2.8 Physiological Relevance

The SHM is able to reproduce several characteristics of complex heart rate dynamics. The first and most obvious effect are fluctuations of the heart rate with the frequency of breathing cycles called respiratory sinus arrhythmia (RSA). This behavior is not surprising as it is directly built into the model with the definition of v_r . A more interesting observation is that the model also exhibits fluctuations with a period of approximately 10 seconds, which also corresponds to a physiological phenomenon called Mayer waves (Seidel and Herzog, 1995). When investigating the reaction of the model to changes in parameter values, Seidel and Herzog (1998) also found bifurcations related to the sympathetic and parasympathetic delays, the baroreceptor sensitivity and repetitive vagal stimulation. The observed dynamical properties were in good agreement with patients with baroreceptor hypersensitivity and animal experiments. However, results by Duggento et al. (2012) show that these bifurcations can actually be triggered by most parameters of the model. They suggest that the model should be reparameterized to make all modeled variables physiologically plausible, and assume that this could lead to a “‘unifying theory to account for slow oscillation’ in cardiovascular variability”.

Kotani et al. (2002) extended the model by noise and more detailed respiratory influences. They showed that the model can explain the synchronization between the heartbeat and the breathing frequency observed in humans (Kotani et al., 2002). In a later study they also found that the (modified) SHM could produce statistically valid simulations of congestive heart failure and primary autonomic failure – diseases that are known to affect the parasympathetic and sympathetic neural activity (Kotani et al., 2005).

To sum up, we can say that the SHM is able to produce physiologically plausible simulations of character-

istics of both healthy patients and several disease conditions. Its potential may not have been fully exploited yet, making our implementation a possible start for future investigations.

3 Implementations and Simulation Setup

Thanks to Henrik Seidel we were able to use his original implementation (SHM-C) as reference for our Modelica implementation (SHM-M). It is written in C and uses a self-implemented Runge-Kutta Method for solving the differential equations. The executable allows to set starting values and parameters with a parameter file and writes the simulation result to a CSV-file.

With our Modelica-version of the Seidel-Herzel-model we wanted to reproduce the output of SHM-C as closely as possible, while still retaining the object-oriented implementation style of Modelica. We divided the formulas of the SHM according to the physiological parts they represent to obtain small Modelica components.

On the highest level SHM-M consists of the models `Baroreceptors`, `SinusNode`, `Heart`, `Lung`, `SympatheticSystem`, `ParasympatheticSystem`, `HormoneRelease` and `NeurotransmitterRelease` as well as the compartment models `BloodSystem`, `HormoneAmount` and `NeurotransmitterAmount`. These models are the entry points for users of SHM-M. Therefore, we kept them as simple as possible by encapsulating the broadening and saturation of the baroreceptors, the contraction model of the heart and the phase effectiveness function into the separate classes `Broaden`, `TanhSaturation`, `Contraction`, and `PhaseEffectiveness`. Additionally, due to the similarities in equation 11 and 12 and equations 16–19, the models `SympatheticSystem` and `ParasympatheticSystem` share a common base class `ANSPart` and the models `HormoneRelease` and `NeurotransmitterRelease` are even functionally equivalent to their base class `SubstanceRelease` only providing different icons. A diagram of the full model can be seen in Figure 1.

Most of the connections between models in the SHM-M are implemented as a set of causal input-/output-connectors. The neural signals and the mechanical respiratory influence as well as the boolean trigger output of the sinus node all have a clear physiological direction. The substance concentrations and the blood flow, however, are implemented using acausal connectors with flow variables. Actually this is not a mathematical requirement, because both the substance concentrations and the blood pressure are defined by a single equation in only one component that could also be implemented with

causal connectors. However, physiologically the release and uptake of substances are separate processes and the blood pressure is influenced by both the Windkessel arteries and the heart itself. For a more realistic representation of these physiological properties, future versions of the model should therefore also separate these effects mathematically, which will be easier to implement using flow variables in the connectors.

With this structure, the implementation could mostly be achieved by just transferring the mathematical formulas directly to Modelica notation. Where approximations of the mathematical definition were necessary – namely the broadening of the baroreceptor response with a Green’s function – the same numerical algorithm used in SHM-C was implemented as a function in Modelica.

The only components where a straightforward implementation was not possible are the sub-models `Contraction` and `Broaden`. The model `Contraction` captures the interplay of the sinus signal, refractory period and AV node and thus features rather complicated expressions in when-conditions involving discrete variables. In the current stable version of the OpenModelica compiler (Version 1.9.1) these discrete equation systems are not supported. We therefore had to implement the contraction signal using continuous variables in the when-condition as the following code snippet shows.

```
when sinus_phase < 1e-10 and signal
  and refrac_countdown <= 0 then
  T_avc = T_avc0 +
    k_av_t * exp(-T_passed/tau_av);
  contraction = sinus_phase > 1 or
    av_phase > 1;
end when;
```

This introduced several additional phase variables for the atrioventricular conduction time, the refractory period and the period of the AV node.

The major challenge regarding the model `Broad` is the implementation of the convolution in Equation 4. We found no better way to calculate this convolution than to build an array with delay expressions of the baroreceptor signal with the following loop:

```
for i in 1:size(hist,1) loop
  hist[i] = delay(x, (i-1)*step, (i-1)*step);
end for;
```

This implementation works as desired for small broadening lengths, but becomes extremely slow for larger values.

As a final difference between SHM-M and SHM-C we did not implement the noise model for the breathing frequency and the heartbeat duration, because this noise would only complicate the comparison of the two models. Instead, to obtain comparable data, noise was also disabled in SHM-C through parameter settings.

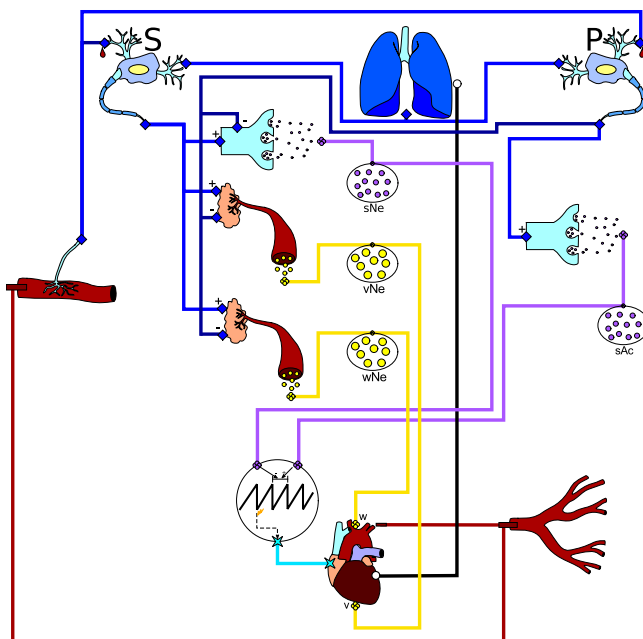


Figure 1. Diagram view of the SHM-M implementation showing the components of the model.

The simulation of our model was performed using OpenModelica. The Runge-Kutta method implemented in OpenModelica uses the same classical RK4-Parameters that are also used in SHM-C. Therefore we only had to use the same step size and the same set of starting values and parameters for the simulation to obtain directly comparable results.

We used the standard parameter set by Seidel and only adjusted the manually defined starting values in SHM-C to match the starting values of SHM-M that were calculated by OpenModelica. The resulting parameter configuration can be seen in Table 1. We then did a simulation with both models for 1000 seconds with a step size of one millisecond and recorded the important values in steps of ten milliseconds as well as the duration and end time for each heartbeat.

4 Results

The SHM was designed for the analysis of heart rate variability. Therefore, to compare different implementations it is most important to look at the duration of heartbeats and the blood pressure. A direct comparison of the time series for these physical quantities can be seen in Figure 2. For the blood pressure both curves have no visual differences until 5 seconds after the start of the simulation, when SHM-M starts to run ahead slightly. At the end of the simulation, the situation is similar: The only difference between the curves seems to be a time shift. For the heartbeat duration the differences are already noticeable at the first heartbeat, which is 3 milliseconds longer in SHM-M than in SHM-C.

To better quantify these differences, we plotted the

Table 1. Parameter and initial values used for the comparison of SHM-C and SHM-M.

Parameter	Value	Parameter	Value
<i>Baroreceptors</i>		<i>Sinus node</i>	
$P_b^{(0)}$	60	$T^{(0)}$	0.9
k_b^{dp}	0.06	k_ϕ^{sNe}	0.6
$P_b^{(c)}$	120	k_ϕ^{sAc}	0.2
σ	0.001	<i>Contraction</i>	
η	0.01	$T_{avc}^{(0)}$	0.09
<i>Lung</i>		k_{avc}^t	0.78
$\theta_r^{(0)}$	0.16	τ_{avc}	0.11
\bar{T}_r	4	T_{refrac}	0.22
$k_{T_r}^{last,1}$	0	T_{av}	1.7
$k_{T_r}^{last,2}$	0	<i>Heart</i>	
σ_{T_r}	0	τ_{sys}	0.125
<i>ANS</i>		C	2
$v_s^{(0)}$	50	$S^{(0)}$	110
k_s^{br}	0.38	k_S^{vNe}	110
v_s^r	30	k_S^m	0
$v_p^{(0)}$	10	$p_w^{(0)}$	0
k_p^{br}	0.38	$\tau_w^{(0)}$	1.3
v_p^r	30	k_w^{rNe}	0.8
<i>Concentrations</i>		\hat{T}	1
k_{sNe}^s	0.014	<i>Initialization</i>	
θ_{sNe}^s	2	$p(0)$	100
k_{sNe}^p	0.006	$c_{sNe}(0)$	0.12
θ_{sNe}^p	0.4	$c_{sAc}(0)$	0.5
τ_{sNe}	2	$c_{vNe}(0)$	0.12
k_{vNe}^s	0.014	$c_{rNe}(0)$	0.12
θ_{vNe}^s	2	T_0	1
k_{vNe}^p	0.006	$t_{c,0}$	-1
θ_{vNe}^p	0.4	$T_{avc}(0)$	0.15
τ_{vNe}	4	S_0	110
k_{rNe}^s	0.014		
θ_{rNe}^s	3		
k_{rNe}^p	0		
θ_{rNe}^p	0.4		
τ_{rNe}	4		
k_{sAc}^p	0.005		
θ_{sAc}^p	0.4		
τ_{sAc}	0.05		

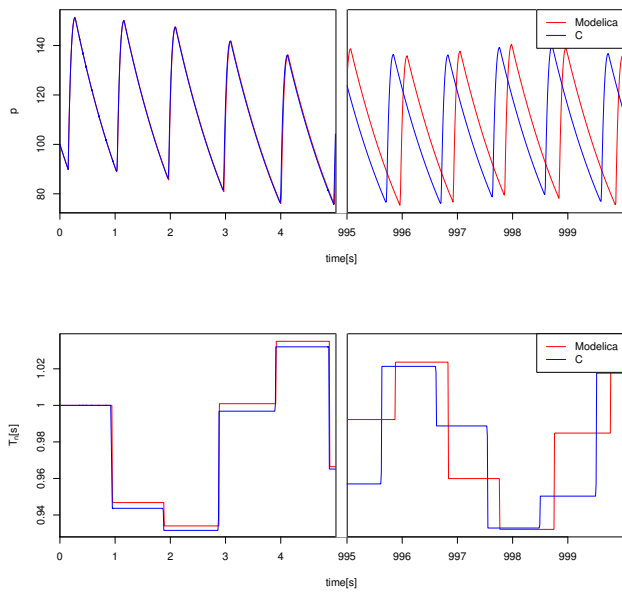


Figure 2. Comparison of time series of blood pressure p and heartbeat duration T_n between SHM-C (blue) and SHM-M (red) for a simulation time of 1000 seconds, showing seconds 0 to 5 and seconds 995 to 1000.

difference between heartbeat durations in SHM-M and SHM-C against the standard deviation between heartbeat durations in SHM-C. The result can be seen in Figure 3. It turns out that the duration of the first 40 heartbeats only differs by less than 10 milliseconds with a standard deviation of 34 milliseconds. The plot also shows that on average SHM-M produces heartbeat periods that are 3 milliseconds longer.

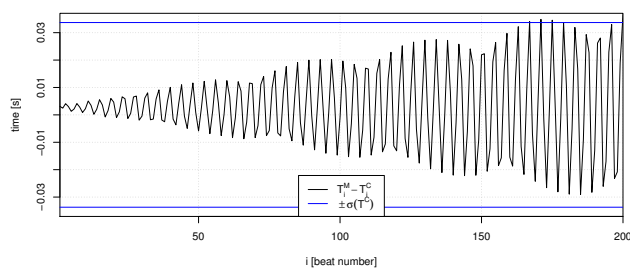


Figure 3. Difference between RR-Intervals of SHM-M and SHM-C relative to the standard deviation of RR-Intervals in SHM-C. Values above zero represent RR-Intervals that are longer in SHM-M compared to SHM-C.

While absolute differences can give an impression of the size of possible calculation errors, for the SHM it is much more interesting to look at the long-time behavior of the model. Seidel used a plot of the spectral density of RR-Intervals (i. e. heartbeat durations) as one of his main arguments for the physiological plausibility of his model. We therefore also compared SHM-C and SHM-M on the

frequency domain. The result can be seen in Figure 4. The plot shows a clear peak identical in magnitude and position at approximately 0.25 Hz, which corresponds to the breathing frequency and can thus be thought to represent respiratory sinus arrhythmia. We can also see another less pronounced peak for both implementations at approximately 0.1 Hz which Seidel attributes to Mayer waves. However, in SHM-M the peak at 0.25 Hz is much sharper than in SHM-C.

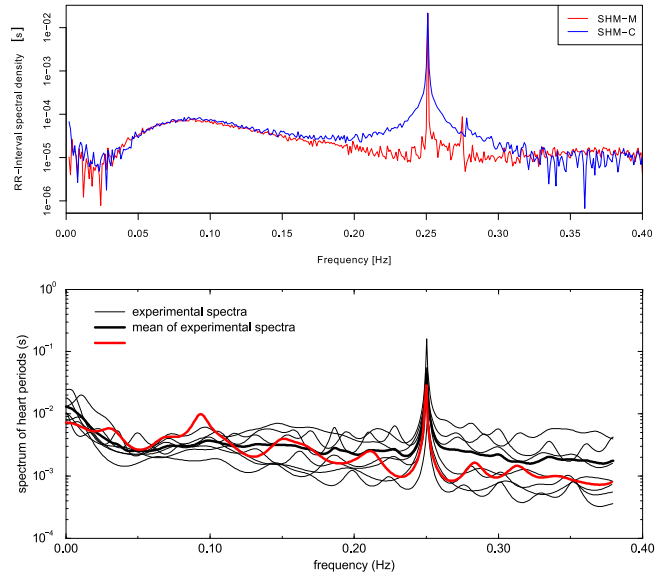


Figure 4. Top: Comparison of spectral density of RR-Intervals in SHM-C (blue) and SHM-M (red) after a simulation for 1000 seconds with standard parameters. Bottom: Figure from Seidel's doctorate thesis comparing spectral density of RR-Intervals of SHM with noise (red) and experimental data (black) (Seidel, 1997).

5 Discussion

The comparison between both implementations SHM-M and SHM-C shows that our proof of concept was successful. At the beginning of the simulation, the blood pressure stays almost the same. There are noticeable differences in the heartbeat duration, but these are not unexpected as the event finding mechanism of Modelica can determine the exact time at which an event occurs more precisely compared to the simple check after each Runge-Kutta step implemented in SHM-C. This is also consistent with the sharper peak in the frequency domain observed in Figure 4. Experimental data with voluntarily controlled breathing actually shows a rather smooth RSA-related peak in the frequency domain similar to SHM-C. However, the reason for this is that the subjects naturally cannot time their breathing to the exact millisecond, introducing a noise to the breathing frequency. This type of noise has been incorporated into the model by Seidel and it can also be incorporated in SHM-M. Our model therefore may allow a more precise

analysis of the theoretical effects of RSA without sacrificing realism.

Now that we have seen that SHM-M is able to reproduce the simulation results of SHM-C we can ask what makes the new implementation (or biological models in general) interesting from a Modelica perspective. First of all, the notion of a system composed of multiple organs fits nicely into the object-oriented paradigm and leads to a natural and intuitive class structure. In fact, each subsection in the explanation of the SHM in section 2 corresponds exactly to a Modelica model in SHM-M. The model structure directly reflects the structure of the real world system and thus makes the model very explainable and accessible for domain experts. Furthermore, encapsulation, inheritance and the reuse of objects instantiated with different parameters could be applied to yield a hierarchical structure that hides implementation complexities and avoids code repetition. The idea of hierarchically structured taxonomies is deeply rooted in biology. It therefore seems reasonable to expect that most biological models can be implemented in such a clean and intuitive manner using Modelica's object-oriented approach.

Additionally, due to the tendency of biological systems to feature multiple complex rhythms, these models can showcase the strength of Modelica's event finding mechanism in comparison to a naive implementation of the Runge-Kutta- or Euler-method or other tools that mainly focus on continuous modeling.

These complex rhythms also turned out to be one of the two major challenges that arose during the implementation of SHM-M. As already mentioned a straightforward implementation of the contraction model was not possible in OpenModelica, because the nontrivial conditions in the when-equations formed a discrete equation system. We did not test the model with other compilers, so this may be only an issue with OpenModelica, but nonetheless our biological model requires a feature that seems not as crucial for most other application areas of Modelica.

The second major implementation challenge was the broadening function used for the baroreceptors. To assess the performance issues with the implementation using direct delay equations, we recorded the time taken for a simulation over 1000 seconds for both SHM-M and SHM-C with broadening lengths ranging from 0.1 seconds to 3 seconds. We found that simulation times of SHM-M rise linearly from 75 seconds with a broadening length of 0.1 seconds to as much as one hour for a broadening length of 3 seconds. In contrast, SHM-C only shows an increase from 16.6 to 21.3 seconds respectively. This results suggest that OpenModelica uses a separate history buffer for each delay equation in the loop. If this is the case, an increase of the broadening length by only one simulation step would require the allocation and management of an additional buffer of the same size as the single history buffer used in SHM-C, explaining the additional overhead. We are not aware of

a language construct that allows to indicate that the delay equations in the loop may share the same buffer. Building the buffer manually in Modelica is also not possible, because the language itself has no notion of discrete simulation steps. This performance issue is therefore hard to fix as a Modelica programmer and would possibly be an argument to include convolutions as a language element.

We can therefore conclude that the SHM, as a model that exhibits the typical properties of biological models in general, does not only fit nicely into the modeling style of Modelica but also has some challenging aspects that point to possible areas of improvement for OpenModelica or Modelica in general. This suggests that biological models could indeed become a new and interesting application area for Modelica.

6 Conclusion

With our implementation of the SHM we demonstrated as a proof of concept that Modelica is perfectly suited for the implementation of biological systems in a natural representation. The language can directly reflect the biological composition of the system instead of having to fit the system into the language constructs. This is shown by the fact that our Modelica version of the SHM – which uses a lot of the features of Modelica such as acausal declarations, encapsulation and component reuse through instantiation and inheritance – can reproduce the same behavior as the original reference implementation in C. Moreover, it is in some parts even more precise thanks to the event finding mechanisms of Modelica.

We also demonstrated that new challenges that are not present in other domains may arise when we model evolved rather than engineered systems. The interaction of complex rhythms together with time-varying signal conduction delays lead to a contraction model that could not be intuitively implemented with the current version of the OpenModelica compiler. Additionally, implementing the behavior that the baroreceptor signal reaches the ANS through many different nervous connections with varying delays required a convolution that seems to be a performance bottleneck.

These are strong arguments both for biologists to choose Modelica over a general purpose programming language and for Modelica modelers to look for interesting applications and models in the systems biology domain. This paper laid the ground for the implementation of more biological models from the side of the Modelica community, but to encourage interdisciplinary research we also have to take the opposite perspective. We need to investigate the benefits of Modelica more closely in regard to the needs of systems biologists. A first step could be to reparameterize the SHM as suggested by Duggento et al. (2012) and to incorporate additional components that can simulate vagal stimulation and different disease conditions (which is possible but

cumbersome with the C implementation). We also plan to extend the SHM to a multi-scale model, for example by exchanging the heart model with a more detailed representation modeling individual heart cells. Finally, it would be interesting to embed the model into the Physiobrary to provide a single point-of-entry for biologists and physicians interested in physiological modeling with Modelica. We believe that there is a lot of potential for interesting projects involving biological models in Modelica and we are looking forward to seeing more of them in the future.

References

- Anja Brunberg and Dirk Abel. Simulation verkoppelter physiologischer Regelkreise mit Hilfe der objektorientierten Modellbibliothek „HumanLib“. In *Automatisierungstechnische Verfahren für die Medizin, 9. Workshop, Tagungsband*, pages 29–30, Zürich, Switzerland, 2010. doi:10.1524/auto.2011.0951.
- Andrea Duggento, Nicola Toschi, and Maria Guerrisi. Modeling of human baroreflex: Considerations on the Seidel–Herzel model. *Fluctuation and Noise Letters*, 11(1): 1240017, 2012. doi:10.1142/S0219477512400172.
- Gernot Ernst. *Heart rate variability*. Springer-Verlag, London, England, 2014. ISBN 978-1-4471-4308-6. doi:10.1007/978-1-4471-4309-3.
- Robin Holliday. Epigenetics: A historical overview. *Epigenetics*, 1(2):76–80, 2006. doi:10.4161/epi.1.2.2762.
- Hermann-Georg Holzhütter, Dirk Drasdo, Tobias Preusser, Jörg Lippert, and Adriano M. Henney. The virtual liver: A multidisciplinary, multilevel challenge for systems biology. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 4(3):221–235, 2012. doi:10.1002/wsbm.1158.
- Peter Hunter, Peter Robbins, and Denis Noble. The IUPS human physiome project. *Pflügers Archiv - European Journal of Physiology*, 445(1):1–9, 2002. doi:10.1007/s00424-002-0890-1.
- Edda Klipp, Wolfram Liebermeister, Christoph Wierling, Axel Kowald, Hans Lehrach, and Ralf Herwig. *Systems biology: A textbook*. Wiley-Blackwell, Hoboken, New Jersey, 2011.
- Kiyoshi Kotani, Kiyoshi Takamasu, Yosef Ashkenazy, H. Stanley, and Yoshiharu Yamamoto. Model for cardiorespiratory synchronization in humans. *Physical Review E*, 65(5): 051923, 2002. doi:10.1103/PhysRevE.65.051923.
- Kiyoshi Kotani, Zbigniew Struzik, Kiyoshi Takamasu, H. Stanley, and Yoshiharu Yamamoto. Model for complex heart rate dynamics in health and diseases. *Physical Review E*, 72(4):041904, 2005. doi:10.1103/PhysRevE.72.041904.
- Andreas Kremling. *Kompendium Systembiologie: Mathematische Modellierung und Modellanalyse*. Vieweg+Teubner, Wiesbaden, Germany, 2012. ISBN 978-3-8348-1907-9.
- Henry Markram. The blue brain project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006. doi:10.1038/nrn1848.
- Marek Mateják, Tomáš Kulhánek, Jan Šilar, Pavol Privitzer, Filip Ježek, and Jiří Kofránek. Physiobrary - Modelica library for physiology. In *Proceedings of the 10th International Modelica Conference*, pages 499–505, Lund, Sweden, 2014. doi:10.3384/ecp14096499.
- Mohsen Naghavi, Haidong Wang, Rafael Lozano, Adrian Davis, Xiaofeng Liang, Maigeng Zhou, and Stein Emil Vollset. Global, regional, and national age-sex specific all-cause and cause-specific mortality for 240 causes of death, 1990–2013: a systematic analysis for the Global Burden of Disease Study 2013. *The Lancet*, 385(9963):117–171, 2014. doi:10.1016/S0140-6736(14)61682-2.
- Emma Larsdotter Nilsson and Peter Fritzson. A metabolic specialization of a general purpose modelica library for biological and biochemical systems. In *Proceedings of the 4th International Modelica Conference*, pages 85–93, Hamburg, Germany, 2005.
- Denis Noble. The rise of computational biology. *Nature Reviews Molecular Cell Biology*, 3(6):459–463, 2002. doi:10.1038/nrm810.
- Chris P. Ponting, Peter L. Oliver, and Wolf Reik. Evolution and functions of long noncoding RNAs. *Cell*, 136(4):629–641, 2009. doi:10.1016/j.cell.2009.02.006.
- R. Ranzinger and William S. York. Glyco-bioinformatics today (august 2011) – solutions and problems. In *Proceedings of the 2nd Beilstein Symposium on Glyco-Bioinformatics*, Potsdam, Germany, 2012.
- Henrik Seidel. *Nonlinear dynamics of physiological rhythms*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 1997.
- Henrik Seidel and Hanspeter Herzel. Modelling heart rate variability due to respiration and baroreflex. In Erik Mosekilde and Ole G. Mouritsen, editors, *Modelling the Dynamics of Biological Systems*, number 65 in Springer Series in Synergetics, pages 205–229. Springer, Berlin Heidelberg, Germany, 1995. ISBN 978-3-642-79292-2.
- Henrik Seidel and Hanspeter Herzel. Bifurcations in a nonlinear model of the baroreceptor-cardiac reflex. *Physica D: Nonlinear Phenomena*, 115(1-2):145–160, 1998. doi:10.1016/S0167-2789(97)00229-7.
- Eberhard O. Voit. *A first course in systems biology*. Garland Science, New York City, New York, 2013. ISBN 978-0-8153-4467-4.

A City Traffic library

Eashan Liyana¹ Simon Lacroux¹ Jean-Baptiste Barbe¹

¹Digital Product Simulation,
La Celle-Saint-Cloud, France,

eashan.liyana@dps-fr.com simon.lacroux@dps-fr.com jean-baptiste.barbe@dps-fr.com

Abstract

Digital Product Simulation (DPS) created a library for the modeling of city traffic. This library is designed for the development and evaluation of control strategies, rendered possible when vehicles are able to communicate between each other and with their infrastructure. CityTraffic library allows for the implementation of control strategies by all of the players acting in an urban environment (e.g. located in vehicles, with a global server computing set points for the vehicles, or with a traffic management system setting speed limits and traffic light cycles).

The library is divided in two parts, macroscopic traffic and microscopic traffic. Macroscopic components are used to describe road networks such as highways whereas microscopic components allow for modeling city traffic where interactions between vehicles and their environment are many.

By using a City Traffic library, cities can decrease the number of traffic jams on their road network, and improve the overall impact of the traffic on the environment.

Keywords: Macroscopic and microscopic road sections, City Traffic, Intersections, Navigation models, Map Creator, Vehicles with their global consumption

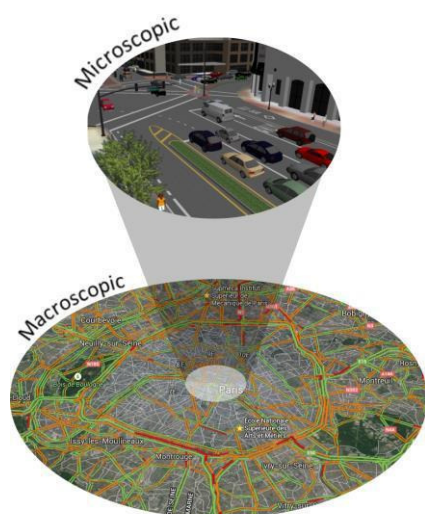


Figure 1: Microscopic and Macroscopic scale

1 Introduction

The management of city traffic and the reduction of motor vehicle emissions are more newsworthy than ever.

As part of the MODRIO project (MOdel DRiven physical systems Operation), a European project financed by the ITEA2 program and lead by EDF, DPS was in charge of the development of a Modeling and Simulation CityTraffic library for a city with two different scales, in order to observe both the flow of vehicles on the road, and the interactions between vehicles and their environment (other vehicles and urban infrastructures).

2 Presentation of the CityTraffic library

The CityTraffic library is divided in two main packages which are the Microscopic scale and the Macroscopic scale. A connection between those scales was implemented as well.

2.1 Macroscopic environment

This modeling scale encompasses an overview of the urban traffic, considering the vehicle flow rate (number of vehicles per hour), not individual vehicles movements. It is based on an analogy with Hydraulics where roads behave as pipes, where intersections are loads, and where vehicles are represented as a fluid.

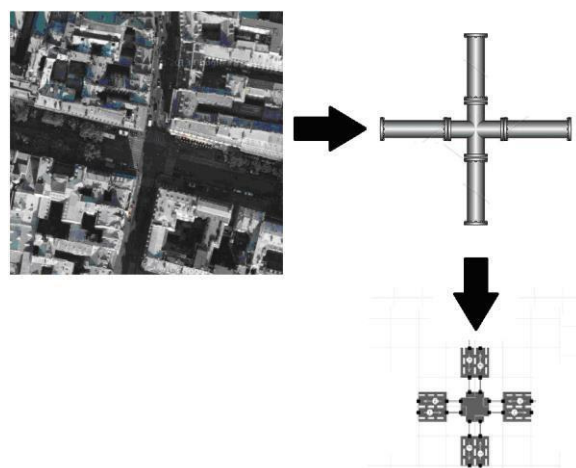


Figure 2: Hydraulic analogy of the macroscopic scale

The macroscopic models are based on the following variables:

- The density of the traffic : $\rho(\text{véh} / \text{km})$
- The velocity of vehicles : $v(\text{km} / \text{h})$
- The vehicle flow rate : $Q(\text{véh} / \text{h}) = \rho \cdot v$

The fundamental equation of the macroscopic environment computes the number of vehicles (Aw & Rascle, 2010) (Iordanova, 2006) (Papageorgiou, 2003)

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho \cdot v)}{\partial x} = 0 \quad (1)$$

Several models were implemented around those variables (first and second order models) with different laws applied to the velocity of the vehicles.

Those models were integrated into components to model roads, intersections or sources of vehicles. The following figure is an example of the components for a road section with two lanes of traffic and two sources of traffic flow thus specifying the number of vehicles generated for the section.

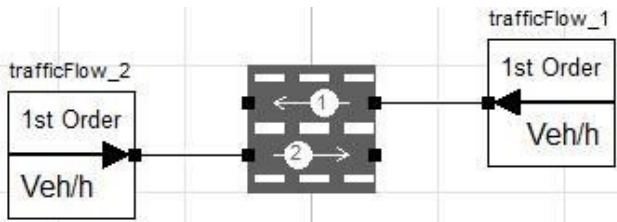


Figure 3: Traffic flow sources for a macroscopic road

Each road is given a maximum density parameter representing the capacity of the road (maximum number of vehicles on the section at the same time). This parameter allows for simulating congestions, that is when the density is higher than this predefined value.

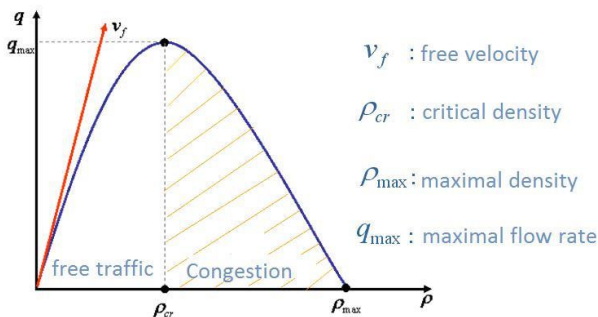


Figure 4: Flow rate as a function of the density of vehicle

This scale gives a really short simulation time, but simulation of interactions between vehicles is impossible. Nevertheless, the macroscopic environment is well suited to modelling highways for

which there is no intersection and where the traffic is more fluid than in cities.

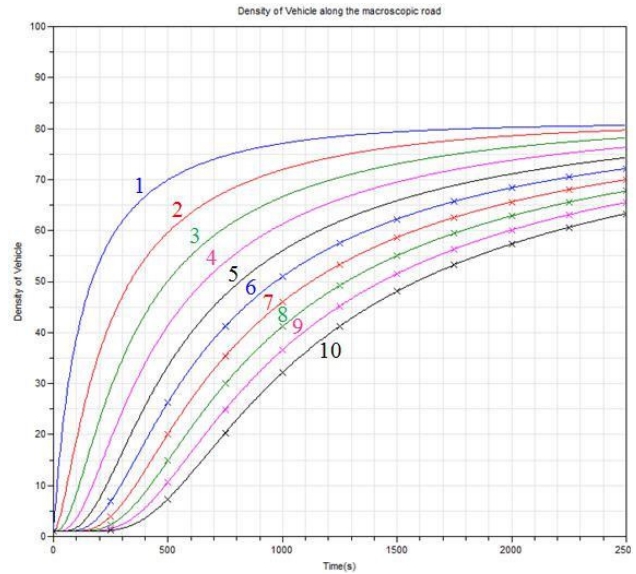


Figure 5: Vehicle density evolution in a 10 km length road over 10 sections of 1km

2.2 Microscopic Environment

This other scale is based on an insider point of view of the urban traffic. It allows for visualizing the journey of each vehicle and modeling the interactions between vehicles as well as between the vehicles and the urban infrastructures (crossroad, red light, stop sign...).

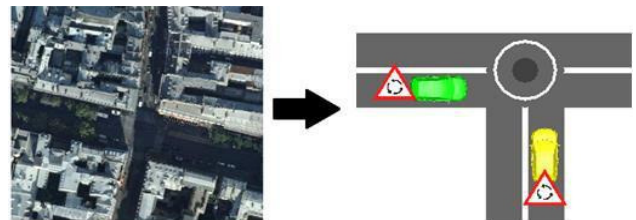


Figure 6: Microscopic scale of a roundabout intersection

The microscopic environment is composed of a Map with a first model named "MapDesign", which is used by a second model named "Env_Micro_and_macro", where the user can define different types of crossroads and initialize the vehicles.

The vehicle model contains a navigation function and a velocity model which are described in part 3.2 of this document. A fuel consumption and a CO2 emission models were added in order to compute the environmental impact of each vehicle.

2.3 Connection between environments

The CityTraffic library includes the possibility to connect the macroscopic and microscopic environments. The simulation time may be reduced by modeling highways or big roads without intersections in the macroscopic scale.

2.4 Performance comparison

In the previous sections we introduced two ways of modeling traffic flow, macroscopic and microscopic. The following figure compares the performance of the two methods.

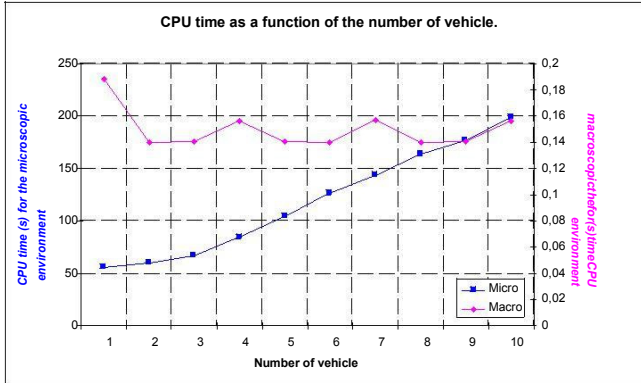


Figure 7: Simulation time as function of the number of vehicles

We simulated a 100m length road for different numbers of vehicle. For one car, the macroscopic model is 300 times faster than the microscopic model, and for 10 cars it's one thousand times faster.

Using the macroscopic method to model areas where interactions are low shortens the simulation time.

3 Content of the library

3.1 Map Generation

The basis of the microscopic environment is the Map where the vehicles move. As for the macroscopic environment for which there is no animation, the map model and the vehicle model are not used.

The Map is composed of nodes whose Cartesian coordinates (x,y,z) are given by the user. Those nodes are then linked with each other, and the lanes of circulation can be defined (one way street or two way traffic). Finally, each road includes a speed limit with a maximum velocity to be observed by the vehicles.

Data of the Map is stored in a spreadsheet (CSV file), which is a matrix with the coordinates of the nodes on the diagonal, and in the other locations the maximum velocity in m/s for the section between two

nodes. In the Map figure 6, nodes 2 (0,100,0) and 1 (0,0,0) are set in both directions with a speed limit of 50 km/h (13.88 m/s). The connections between nodes are modelled with the value of the velocity: between node 1 and node 2, the velocity is in the first row and second column; the second row and first column corresponding with the velocity between node 2 and node 1.

On the contrary, nodes 3 (0,200,0) and 1 (0,0,0) are not connected, and the value for the maximum velocity between them is 0 m/s.

If a road is a one way street, the velocity between nodes will be equal to the maximum velocity for the right way of traffic and will be set at 0 m/s for the other way. For example, between node 5 and node 4, the traffic goes from 4 => 5 (in the row of node 4, there is a value of 13.88 m/s in the column of node 5), but from 5 => 4 is the wrong way and in the row of node 5 there is 0 m/s in the column of node 4 (in red in the figure).

In addition to the CSV file generation, the MapDesign model also creates an animation of the Map where each section is numbered. Roads with a double way of circulation are larger than one way streets, and have two numbers (one for each way of circulation). When there are two numbers, the smaller represents the circulation from the node with the smallest number to the node with the biggest number.

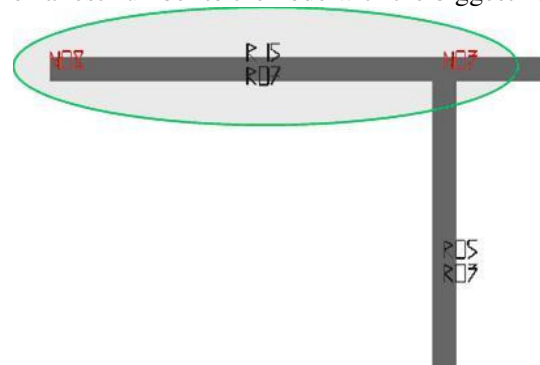


Figure 8: Numbering of the Map sections

For example, on figure 8, section R7 is the road between nodes 3 and 8 for the direction N3 => N8, whereas R15 corresponds to the direction N8 => N3.

0,0,0	13.88	0	0	0	0	0	0	0
13.88	0,100,0	13.88	0	13.88	0	0	0	0
0	13.88	0,200,0	0	0	13.88	0	13.88	0
0	0	0	100,0,0	13.88	0	0	0	0
0	13.88	0	0	100,100,0	13.88	13.88	0	0
0	0	13.88	0	0	100,200,0	0	0	13.88
0	0	0	0	13.88	0	200,100,0	0	0
0	0	13.88	0	0	0	0	-100,200,0	0
0	0	0	0	0	13.88	0	0	200,200,0

Figure 9: CSV File of a Map with 9 nodes

3.2 Vehicle model

3.2.1 Velocity model

Three velocity models were created in the CityTraffic library. The first one is named "Simple" and does not take into account the other vehicles. It defines only a constant velocity. The other two models (Krauss and Gipps) are car-following models (Krauß, 1998), meaning the velocity of the selected vehicle is associated with the velocity of the vehicle ahead. If there is no vehicle ahead, the velocity of the vehicle is linked with the velocity of a virtual vehicle located far away, and it accelerates until the velocity reaches the maximum value allowed on the current road section.

Unfortunately, the Krauss model is not realistic, because acceleration and deceleration are instant instead of being gradual. Finally a better model was created: the Gipps model.

This model is used in many traffic modeling software, and includes a progressive acceleration and deceleration with programmable maximum values. Nevertheless, Gipps corresponds to a perfect driver who respects the Highway Code and the security distance between vehicles. So it does not permit to model an overtaking because the velocity of the vehicle will always be inferior or equal to the velocity of the vehicle ahead. Future development will solve this issue.

For roads with two ways of circulation, a function was created to identify vehicles coming from the opposite way. Then, a passing vehicle will not make the vehicle brake, because it will not consider it as a vehicle ahead.

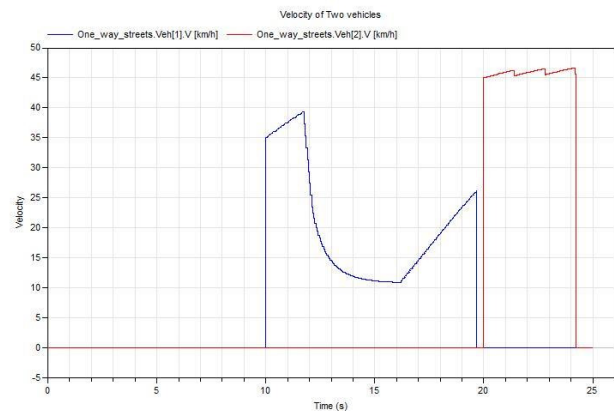


Figure 10: Simulation of two vehicles. The blue car decelerates to turn then accelerates until reaching its destination. The red car is on a straight way and its velocity is limited by the road's speed limit.

3.2.2 Navigation function

The navigation function of the vehicle is based on the "Dijkstra algorithm" (Dijkstra, 1959) which identifies nodes to cross, in order to reach the destination using the shortest way in terms of distance. This algorithm uses the following methodology:

From one node, the function will first determine the nodes which are connected to it.

If a node is not connected to the first one, its "cost" will be set to the infinite value, whereas the connected nodes will have a cost proportional to the distance between the nodes. Then the same process is repeated for all the nodes of the Map until the destination node is reached.

Finally, the nodes are classified by their cost, and the table of nodes returned as output will be a table of the nodes with a minimal cost.

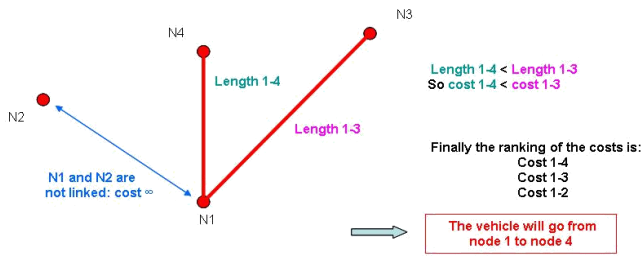


Figure 11: Principle of the Dijkstra algorithm

3.2.3 Types of vehicle

In order to be able to visualize the vehicles on the Map, CAD models are imported for each type of vehicle.

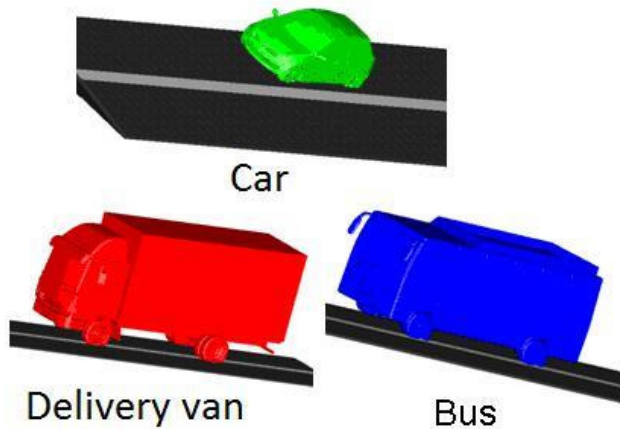


Figure 12: Types of vehicles available

Each type of vehicle has its own characteristics (dimensions, mass...) which are used to update the safety distance between vehicles and to compute the fuel consumption of each vehicle.

3.2.4 Fuel consumption and CO₂ emissions

The fuel consumption model is based on the computing of the traction force (Guzzella & Sciarretta, 2007) (Eriksson, 2013) (SETRA, 2009). In order for the velocity and the acceleration to be constant during a short interval of time, a time stamp is used when computing all of the variables. Thus the traction power is evaluated using a quasi-static method.

With the power calculated, the fuel consumed power "P_f" is computed using the efficiencies of the elements of the transmission.

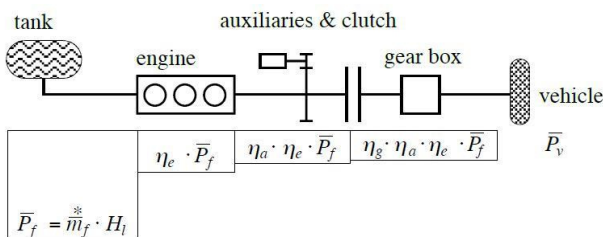


Figure 13: Transmission chain of a vehicle

The engine efficiency is the most difficult to compute, because it depends on the rotational speed of the engine and the ratio of the gear box.

The equation for the flow of fuel is:

$$\dot{V}_f^* = \bar{P}_f / (H_l \cdot \rho_f) \quad (2)$$

With for the diesel:

- H_l = 43.5 *10⁶ J/kg for the fuel's lower heating value
- ρ_f = 0.75 kg/L for the density.

CO₂ emissions are calculated with the following equation:

$$\text{CO}_2 \text{ emission} = \text{Fuel consumption} * \rho_f * H_l * \text{fuel emission factor} * \text{partial combustion factor} \quad (3)$$

These models are included in the vehicle model to determine the exact quantity of fuel consumed and CO₂ emitted during the trip of the vehicle on the Map.

4 Intersection models

Five models of intersection were created (roundabout, traffic light, stop sign, yield and right of way), and three models of stops (bus station, delivery area and pedestrian crossing). Those models can be activated only for the microscopic environment via an interface where the user defines locations of the intersections and stops on the Map by specifying nodes and sections of the intersection.

4.1 Ghost vehicles

When a vehicle "A" reaches an intersection, a "ghost vehicle" is set in its place with a 0 m/s velocity. These ghost vehicles are present on the map but not shown on the animation by Dymola.

As the velocity model detects the presence of the vehicle ahead, the vehicle "A" will adapt its velocity to the ghost vehicle and will stop. When the intersection model has checked that all conditions are fulfilled to let the vehicle go forward, the ghost vehicle disappears and the vehicle "A" may leave the intersection.

Figure 14 is an example of this situation at an intersection. There is a yield at node 2. The road section impacted by the yield is the blue one.

The yellow vehicle has to give way to the green vehicles coming from its left and its right, so the intersection model will identify the black road sections as having priority, and a ghost vehicle will appear in front of the yellow vehicle (in pink on the previous figure). The position of this ghost vehicle will be fixed, and its velocity equal to 0 m/s, so the yellow vehicle will detect it and will stop just behind it.

When green vehicles have passed, the intersection model detects that there is no more vehicles on the priority roads and the ghost vehicle is removed. As the yellow vehicle has no more vehicles ahead, it resumes its trip.

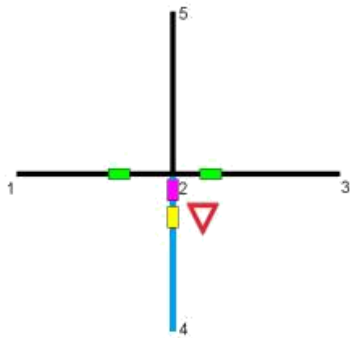


Figure 14: Example of a yield

When the green vehicles have passed, the intersection model detects that there is no more vehicles on the priority roads and the ghost vehicle is removed. As the yellow vehicle has no more vehicles ahead, it resumes its trip.

The following figure shows the animation of an intersection.

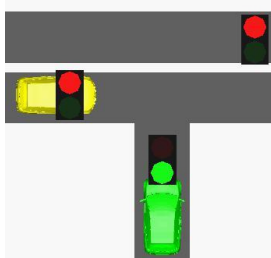


Figure 15: Animation of a red light

4.2 Stops

Other models were implemented to stop the vehicles which are not located at the intersection but in the middle of the road. Those models are the bus stations, the delivery places and the pedestrian crossings.

The first two models impact only one type of vehicle (bus or delivery vans), but the pedestrian crossing model stops all vehicles when pedestrians are crossing.

5 Conclusion

We presented the CityTraffic library which offers an intuitive tool for modeling interactions between vehicles and their environment. The macroscopic and microscopic scales speed up the simulation while computing the impact of an intersection on the traffic flow and the fuel consumption of a vehicle during its trip.

Other intersection models are being created, as well as a more realistic model for the driver with heterogeneous behaviors.

Acknowledgements

The French Ministry DGE has funded this work within the ITEA2 project MODRIO.

References

A. Aw and M. Rasclé (2010): *Resurrection of "second order" models of traffic flow.*

V. Iordanova (2006): *Contribution à la modélisation et la commande du trafic routier : Approches par Bond Graph et commande par platitude.*

M.Papageorgiou (2003): *Review of Road Traffic Control Strategies.*

Stefan Krauß (1998): *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics.*

<http://licence-math.univ-lyon1.fr/lib/exe/fetch.php?media=gladijkstra.pdf>

Lino Guzzella and Antonio Sciarretta (2007): *Vehicle Propulsion Systems, Introduction to Modeling and Optimization.*

Lars Eriksson, Linköping University (2013): *Vehicle Propulsion Systems, Course Introduction & Energy System Overview.*

SETRA (service d'étude sur les transports, les routes et leurs aménagements) (2009): *Etude des émissions routières de polluants atmosphériques.*

An open toolchain for generating Modelica code from Building Information Models

Matthis Thorade¹ Jörg Rädler¹ Peter Remmen² Tobias Maile³ Reinhard Wimmer³ Jun Cao³
Moritz Lauster² Christoph Nytsch-Geusen¹ Dirk Müller² Christoph van Treeck³

¹Berlin University of the Arts (UdK), {m.thorade, jraedler, nytsch}@udk-berlin.de

²RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Aachen, Germany {premmen, mlauster, dmuellder}@eonerc.rwth-aachen.de

³RWTH Aachen University, Institute of Energy Efficient Building, Aachen, Germany

{maile, wimmer, cao, treeck}@e3d.rwth-aachen.de

Abstract

Building Performance Simulation (BPS) is a key element in the design of energy efficient buildings, and there is increasing interest in using the Modelica modelling language for BPS. The IEA-EBC coordinates development of BPS in Modelica in the project “Computational Tools for Building and Community Energy Systems” (Annex 60). However, developing BPS models and collecting required input data are time-consuming and error-prone processes. Reusing existing Building Information Models (BIM) as basis for Building Performance Simulation (BPS) has the potential to make BPS model development and subsequent simulation easier, faster and more reliable. Activity 1.3 of the Annex 60 project is working on an open-source toolchain that can semi-automatically generate code for BPS Modelica models from a BIM data source. Parts of that toolchain are discussed in this paper.

Keywords: Building Information Modelling, Modelica code generation, Building Performance Simulation

1 Introduction

Buildings become increasingly integrated to reduce energy and peak power and to increase occupant health and productivity, leading to complex building design. Building Performance Simulation (BPS) is one key element in the design of energy efficient buildings. The Energy in Buildings and Communities Programme (EBC) of the International Energy Agency (IEA) launched in 2012 the project “Computational Tools for Building and Community Energy Systems”, also known as Annex 60 (Wetter and van Treeck, 2012). The Annex 60 project aims at developing next generation computing tools for the buildings industry, based on open non-proprietary standards, including the Modelica modelling language and the Functional Mockup Interface. The structure and

organization of the project into subtasks and activities is shown in Figure 1.

The development of Modelica model libraries for BPS before Annex 60 was fragmented with the result that each institution was developing the same components in a different, possibly incompatible, manner. Activity 1.1 focuses on harmonizing BPS library development by providing a core library of base classes and components commonly needed. The libraries currently contributing to and relying on the Annex 60 core library are:

- AixLib from RWTH Aachen (Fuchs et al., 2015)
- BuildingSystems from UdK Berlin (Nytsch-Geusen et al., 2013)
- Buildings from LBNL (Wetter et al., 2014)
- OpenIDEAS from KU Leuven (Baetens et al., 2015)

Each library extends the core library by providing additional components for special applications, depending on the respective institutions research focus. The libraries AixLib and BuildingSystems will be used later in this paper to demonstrate the code generation.

But even with advanced component libraries available, building up BPS models from hand and collecting required input data remain time-consuming and error-prone processes (Bazjanac et al., 2011), preventing practitioners from using BPS more extensively in standard planning processes. Building Information Modelling (BIM) is a well established technology to model and manage the digital representation of a building over its entire lifecycle (see e.g. Eastman et al., 2008). Reusing existing Building Information Models (BIM) as the basis for Building Performance Simulation (BPS) has the potential to make BPS model development and subsequent simulation easier, faster and more reliable.

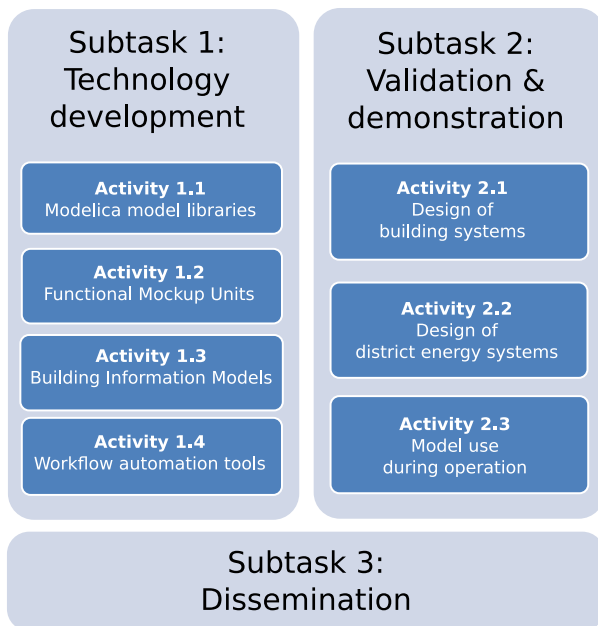


Figure 1. Structure and organization of the IEA EBC Annex 60 (figure adapted from Wetter and van Treeck (2012)). In this paper, results from Activity 1.3 are presented.

Activity 1.3 of the Annex 60 project aims at leveraging such BIM systems as a source of information for semi-automated generation of BPS models in the form of Modelica code. In order to reach a broad audience, the methods and tools developed should:

- make use of, comply with and contribute to existing open standards,
- support both building geometry as well as heating, ventilation and air conditioning (HVAC) components,
- support multiple Modelica libraries,
- support a high degree of automation,
- be a collection of small, reusable tools,
- be released under open-source licenses.

A description of the toolchain implementing the overall process as planned in this project is given by Remmen et al. (2015), including a review of prior work in the field. That paper is also summarized in the following section, giving an overview of the overall process, basic ideas, assumptions and software foundation. The section Python Framework then describes in more detail the part of the toolchain that is used for controlling the workflow, via a GUI or via Python scripts, as well as the actual code generation. The section Use Cases gives a first, simplified demonstration of the process. The paper then concludes with a short discussion of limitations and future work.

2 Process Overview

The whole process of generating Modelica code from Building Information Models is in this project implemented as a toolchain of various special-purpose tools. Having various tools with a clearly defined task means these tools can be developed partly independent, and each block can possibly be reused in a different context. On the other hand, interfaces between the tools have to be clearly defined, either in the form of a file format or as an Application Programming Interface (API). The various steps of the process and the interfaces are shown schematically in Figure 2 and are discussed in the following paragraphs, summarizing the paper by Remmen et al. (2015).

Creating the Building Information Model A typical starting point for a BIM-based workflow would be an architect creating a designated space and usage structures using a BIM-based CAD software. Other domain experts, e.g. HVAC engineers, then enrich the BIM by contributing further data. In order to collaboratively create the model, all involved actors use a common file format. IFC is a well-established, non-proprietary and standardized BIM file format (International Organization for Standardization, 2013). In this project, we rely on version 4 of IFC, because it contains several improvements over its predecessor IFC 2x3. Using a standard format means that various applications on the market will be able to deliver input to our toolchain. Also, we profit from existing tools for checking, viewing or sanitizing IFC files.

Transformation to Simulation Domain Model While IFC is a well-established BIM file format, it is in its current form not very well suited as direct input for Building Performance Simulation (BPS), because it does not contain all information required for BPS (e.g., some detailed HVAC objects and properties are missing, as well as simulation specific objects and properties), and it has rather long turnaround time for changes. Information models for the simulation domain and corresponding file formats have been developed with the goal to resolve these drawbacks. In this project, we rely on SimModel and corresponding SimXML files as defined by O'Donnell et al. (2011). The SimXML file format is clearly defined by an XML Schema Definition (XSD). SimModel closely aligns with IFC regarding building geometry and building physics, but removes some redundancies and simplifies relationships between objects. Besides the IFC model it also entails other datamodels such as gbXML and others. Regarding HVAC components, SimModel is a superset of IFC. That means missing information has to be added during the transformation process. For conversion of geometry and building physics data ex-

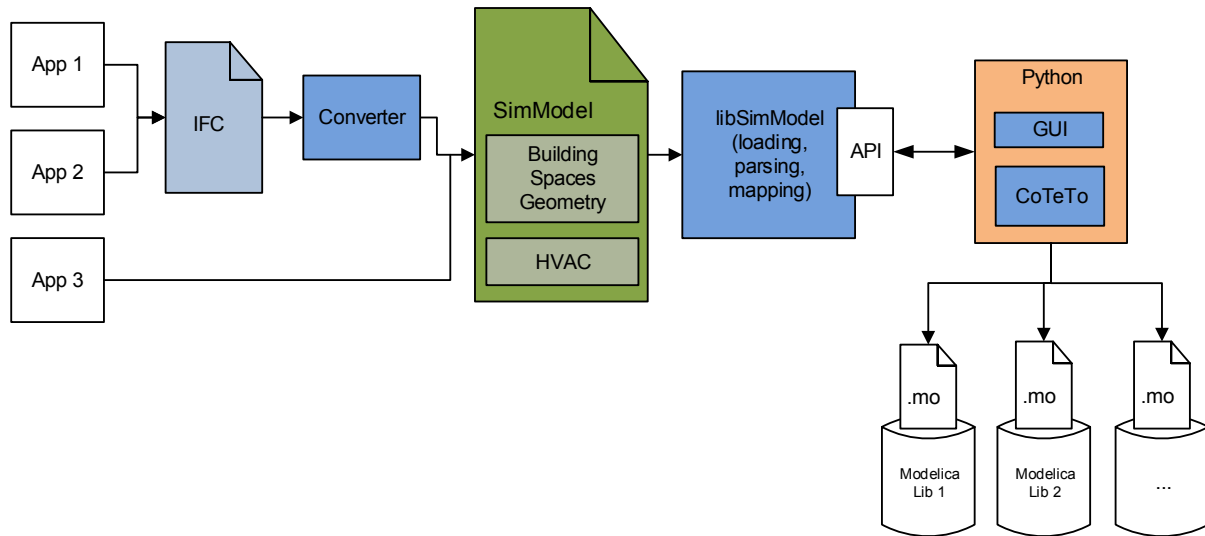


Figure 2. EnEff-BIM Process Overview.

isting tools can be used, such as Simergy and/or Space Boundary Tool (SBT) (Rose and Bazjanac, 2015). For conversion of HVAC data a new tool is developed in this project. The two converted parts have to be combined in a second step into a single valid and correct SimXML file that is information complete, as expected by the following steps in the overall process. Another option that is also investigated in this project is to create (parts of) SimModel files with only few high level input parameters provided by the user (such as year built, type of building, etc.) and filled with typical and statistical data for a complete model.

Mapping to Modelica libraries To accomplish the link between the rigid data structure in SimModel with the flexible data representation on the Modelica side, mapping rules are needed (Wimmer et al., 2014). Loading the SimModel file, parsing it and mapping the data from SimModel to Modelica is performed by the C++ library libSimModel, described in detail by Cao et al. (2014, 2015). The SimXML file is first loaded by a validating parser that uses the XSD. As part of the parsing process, a hierarchical tree is built up and some manipulations and simplifications, like resolving links, are performed. The data, once loaded, is then mapped using the library specific mapping rules as described by Wimmer et al. (2015). These mapping rules are valid for a specific version of a specific library. When the library changes, the corresponding mapping rules have to be updated. The mapping rules are again stored in XML files (confirming to a corresponding XSD). To ease maintenance of the mapping rules, a tool for conversion between a spreadsheet table and the corresponding mapping rule XML file is developed.

All mapped data and, as needed, also unmapped data, as well as the methods and functions of the libSimModel library for loading, parsing and mapping are exposed to Python as an API.

Code Generation and User Interface The last part of the toolchain is written in Python and it covers three tasks: Process control, Information Pre-Processing and the actual Modelica code generation. These tasks and the implementation are discussed in the following section.

3 Python Tools

The Python tools cover three tasks: Process control, Information Pre-Processing and the actual Modelica code generation. The organization of the tools is shown in Figure 3.

3.1 Process Control

As discussed in the previous section, the whole process is implemented by several components that build a toolchain. For the normal end-user this chain should appear as one tool, but for power-users and during development all parts should be usable standalone. To achieve this, some requirements must be fulfilled:

- a common programming platform (language, versions),
- an Application Programming Interface (API) to call the components functions from the common programming platform,

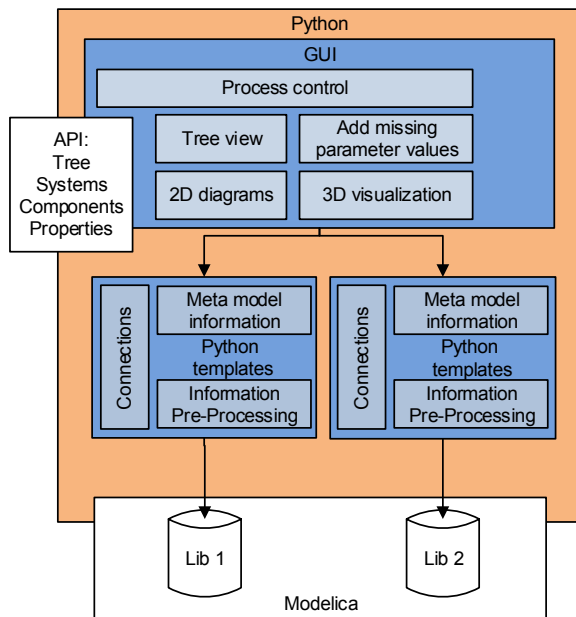


Figure 3. The Python part of our toolchain.

- no own or just an optional or partial graphical user interface (GUI).

These requirements are already fulfilled for that part of the toolchain that use SimModel as a starting point. That part will be controllable from the common process control using the GUI or scripts, standalone or embedded in other tools. Further parts of the toolchain will be added to the common process control as needed.

As the common platform for integrating the components Python is used, so all components must be usable from this language. This requirement is fulfilled if the component is already written in Python, otherwise a wrapping API is necessary. Python is well suited to bind different components to one tool, and it is widely used and accepted in the Modelica community. The GUI part is implemented in Qt, so it will be portable to all major operating systems.

3.2 Information Pre-Processing

While SimModel was designed to contain all information that is required for BPS, that information may sometimes have to be processed or converted. Simple conversions (e.g. unit conversion) will be covered by the mapping rules, but more complex conversions are more easily implemented in Python. One example for complex information conversion is the calculation of equivalent lumped heat capacities according to VDI 6007 that are used in low-order models of AixLib (German Association of Engineers, 2012).

The mapping rules also do not cover the connections between objects, this information is passed to Python as meta model information. Another task is the process-

ing of Modelica graphical annotations. This is work in progress and will be extended as needed. The information processing is implemented as Python filters for the templates, as described in the following section.

3.3 Modelica Code Generation

The actual Modelica code generation is implemented as a tool named CoTeTo, which stands for Code Templating Tool. Although designed for this project, this tool was implemented in a way that it can be used standalone and in other software environments. CoTeTo will be released under an open-source license.

In this project, Modelica models for a set of different model libraries have to be generated using a common data source. Each library needs separate filtering and output of data because of different modelling approaches. These libraries are currently under development and are likely to change in the future as well. This requires a flexible and generic data conversion framework to allow for future changes. Thus, the framework should allow flexible output components for different libraries in multiple versions as well as flexible input components, both should be easy to maintain even for non-programmers. The workflow of CoTeTo and the coupling to other tools within the toolchain is shown in Figure 3. We designed CoTeTo to be used by graphical, command line and library level interfaces. The multiple access possibilities open the framework to a huge community. The fact that Python does not require extensive compilation cycles helps with rapid development. The following section will give an overview of the components and their functionality. We have divided CoTeTo into input components (*Data APIs*) and output components (*Generators*). A *Generator* depends on a specific *Data API* (defined by its name and version).

The Template Approach There are two general concepts for the generation of textual output within a computer program. One approach is to embed `print()`-statements for text strings and data in the structure of a program. This is useful for nearly static, well-defined structures of the data set and of the textual output.

The other approach is template-based, where placeholders for the content are embedded in a text file (a template for the output). Besides placeholders templates also offer control structures. Thus, template-based model generation allows complying with fixed Modelica language syntax and adding flexible model content in the same file. One advantage is the flexibility for the end user, who does not necessarily need to dive into the programs' internal structure, but can just enrich the template file with placeholders and simple programming constructs, whenever the used Modelica models change. This workflow is much like the form letter function in office software, which fills some variable address fields in a text document from a database.

The template approach fits very well into the flexible structure of the CoTeTo framework, as it is independently usable for different information sources. From the list of available template engines Mako (Bayer, 2014) and Jinja2 (Ronacher, 2014) seem to fit best into CoTeTo. At this point support for both is implemented, but after an evaluation phase one of the engines may be dropped in the future.

Input - Data APIs A *Data API* is a Python module that defines a prescribed way to fetch data sets from a data source. Although we use the Python language to write the CoTeTo, *Data API* functions can interface to other languages.

Different *Data APIs* and different versions can be used in parallel. Sample modules for reading JSON, XML and CSV files exist in CoTeTo. This allows flexible processes during development and testing. There is no definition for the structure of the returned data items, since different data sources contain different types of data (tree, table, graph, map). It is the job of the used output *Generator* to understand the data delivered by the used *Data API*.

The most important *Data API* in the Annex 60 context is the interface to the C++ library libSimModel, which handles the SimModel parsing and the mapping to Modelica. Seen from the Python framework and from CoTeTo it defines the data source used to fill the placeholders in the output templates.

Output - Generators Once all relevant data has been loaded into CoTeTo, it is passed to the output component, called *Generator*. We designed the *Generator* to contain all items needed to generate the code for a specific Modelica library. This includes

- filter functions,
- the meta model structure,
- text templates,
- additional configuration and information and
- additional files.

The filter functions, meta model structure and text templates are used and applied by CoTeTo. Additional files like the mapping rules XML file can be stored inside the *Generator*.

We experienced that some data need manipulation that may not fit well into the mapping rule mechanism. For this purpose, *Generators* can include filter functions (Python code) that we call between the data API and the templates. The filters are custom-built to the used library. In our case, they may include simplification of geometric relationships and calculation of

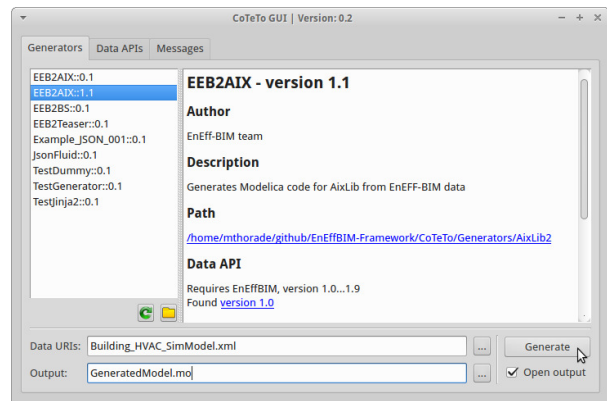


Figure 4. Standalone GUI of CoTeTo

model specific parameters. Another application of filters would be the creation of annotations for the graphical appearance and placement of model components in the Modelica code.

One major challenge in the automated generation of Modelica models is the flexibility of Modelica. Generally said, setting up useful models needs the knowledge of an experienced user. We are following the approach to encapsulate this knowledge in library specific meta-models and templates. One essential task is the appropriate connection of components. The API returns the connection information corresponding to the SimModel ontology, which differs from the one in a Modelica library. The meta model checks if the connection is applicable, if not, it manipulates it.

The text templates are the last step in the process chain. The template engine combines the data structures returned by the *Data API* and possibly manipulated by filters with the text templates to files with valid Modelica code. The templates in a *Generator* can be splitted into several files to ease maintenance.

Generators can be easily exchanged between different installations, as they are simple folders or even zip-files with a defined structure. *Generators* can be maintained and edited with standard system tools like a file manager and a text editor. Creating a new *Generator* is as simple as copying a folder with an existing *Generator* and changing the name or version number in a text file.

Interface and Handling There are currently three ways to use CoTeTo:

- CoTeTo can be imported in Python software as a module library. CoTeTo works both with Python 2.7 and 3.3+. All functions are usable via the modules API.
- A command line interface can be used interactively or called from other software. It allows listing the available *Data APIs* and *Generators*

and executing a *Generator* with a data source URI to produce the text output.

- The graphical user interface (GUI) is implemented using PyQt4. It allows flexible browsing and editing of all components and included files and the execution of selected *Generators*. The GUI can be used as a standalone tool (see Figure 4) or embedded in PyQt4-based applications as a widget.

4 Use Cases

To prove the concept of the toolchain, we have developed several use cases. These use cases are kept simple to ensure the focus on the process. Each use case consists of one single room, according to the description of the validation example of German Guideline VDI 6007 (German Association of Engineers, 2012) and varying HVAC setups. We divide the use cases in two groups, water- and air-based systems. The water-based systems are only meant for heating, while the air based systems also cool and ventilate the room.

The first group of use cases has some basic elements in common. These include a pump, pipes, a PID controlled valve and an expansion vessel. The efficiency of the pump is given depending on the volume flow. The control strategy of the pump includes a night set back, where the volume flow is reduced during nights. We designed the use cases to be controlled by a PID controlled valve. The control variable is the room temperature. Besides these common components the water-based systems differ in the heat generation and heat distribution. The different combinations are as follows:

- 1.1 Boiler & Radiator
- 1.2 Boiler & Radiator & Domestic Hot Water System
- 2.1 Heat Pump & Radiator
- 2.2 Heat Pump & Floor heating
- 3.1 CHP & Boiler & Radiator

The second group of use cases consist of two air-based setups. Similar to the first group they have most of the components in common, like air ducts, fans, filter, damper and silencer to account for additional pressure losses. They differ in the purpose of the ventilation system. The first of the two use cases is primarily heating and includes an electrical heater, the other use case is primarily cooling and includes an evaporative, adiabatic cooling device.

The following section presents the first use case (use case 1.1) and the corresponding Modelica code generation using `AixLib` and `BuildingSystems` library. As we focus on the HVAC system, we will describe the code generation for this part of the model only. The thermal zone is currently modeled using the low order model from `AixLib` for both implementations. The hydraulic schema is shown in Figure 5. A gas boiler

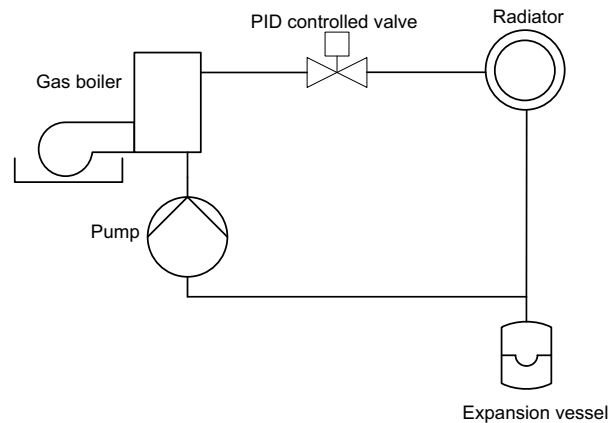


Figure 5. Hydraulic schema of the applied use case

heats the water to a fixed set point temperature. A pump circulates water in the hydraulic system. A radiator emits the heat to the thermal zone. The parametrization of the radiator follows the DIN-EN 442 (DIN German Institute for Standardization, 2015). We designed the radiator to be controlled by a PID controlled valve. The use case is completed by connecting pipes and an expansion vessel.

By calling the API, we load the contained data from the SimXML file into our Python framework. This data is already mapped to the corresponding library as previously described, in our case to `AixLib` or `BuildingSystems`. In addition to the mapped data, the API returns the topology of the use case as a sorted list. The next step is to check if the SimModel topology fits with the Modelica topology in terms of positioning of the components according to the hydraulic schema and correct Modelica connections. By analyzing the Modelica models we identify four different connection types we have to handle in the use case, all types are included in the Modelica Standard Library (MSL). These four connections types correspond to the SimModel connections. The connections (MSL) and their relations to SimModel are as follows:

Table 1. Comparison of Modelica and SimModel connectors

MSL	SimModel
Fluid connector	Water connector (cold, hot)
Thermal connector	Air connector
Real connector	Control connector
Boolean connector	Control connector

The framework connects one component after another according to the given topology, ensuring that the connectors of the models match. In our first use case we have a simple loop and all components, except the expansion vessel, extend from a simple two port model. This is a straight forward approach, as the topology between the hydraulic schema, SimModel and

Modelica does not differ much. Components that are not considered in SimModel, like the expansion vessel are automatically implemented in every hydraulic loop modeled in Modelica. The meta model contains information about the connection of each model and information about components that need to be inserted automatically. The model for a thermal zone from AixLib has two Thermal connectors for the implementation of convective and radiative heat sources. Both radiators from the two libraries also have a convective and a radiative thermal connector, while SimModel uses a single air connector. Further, the API passes the information that the radiator and the thermal zone are coupled. The meta model collects and compares all this information and produces a connection between radiator and thermal zone.

More challenging is the correct choice of control systems and their connection to the components. The chosen controller set up in Modelica depends heavily on the used component model. For example, the pump model in AixLib has a Boolean input that can turn on and off a night mode with a reduced volume flow. This allows a direct use of `Modelica.Blocks.Sources.BooleanPulse` as a control element. The pump of BuildingSystems requires the pressure rise over the pump as a Real input. This example shows possible differences in Modelica's control implementations. Typical control strategies, like a night set back, are embedded as templates in the meta model. These strategies are directly mapped to the ones in SimModel.

Once all data is processed, the result is a valid Modelica model. A Modelica representation of this specific use case is given in Figure 6, here using the BuildingSystems library. The mentioned pump control is highlighted in red. At this stage of the project the graphical layout of components in Dymola is not supported and needs manual input.

5 Summary

5.1 Limitations

As Annex 60 is an ongoing project and all tools are currently under development, we are aware of limitations in the discussed framework. Some of the limitations will be tackled in ongoing work, others are out of the projects scope. The following section provides an overview of known limitations. As the Building Information Model comes in the form of an IFC file, we assume a valid, well formed model. The IFC file is the foundation of the presented toolchain. For example the IFC file has to contain SpaceBoundary entities. Yet, the process is semi-automated and still needs input from the user. Whenever the Modelica libraries change, the

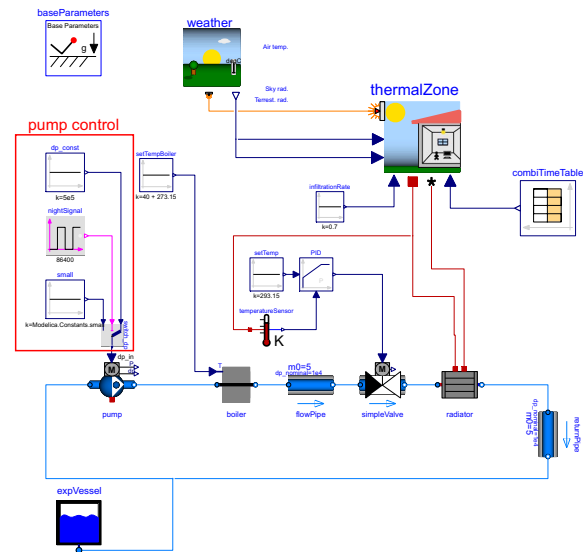


Figure 6. One of several use cases (here shown with BuildingSystems).

mapping rules and in some cases also the templates in CoTeTo have to be adapted. For this reason we enable the use of different versions of Modelica library and the corresponding mapping rules and templates. Although the latest versions of the libraries are used, the initialization of the Modelica model may not provide good starting values. The model still needs fine tuning and the correct choice of initialization values. The generated Modelica model can be seen as a first starting point. This is also true for the graphical arrangement in the used simulation environment, in our case Dymola. For simple models like the Use Case, the representation is fairly straight forward. If we look at more complex systems a meaningful arrangement is more challenging and currently not supported. Further limitations are that currently only two libraries of the Annex 60 project are supported and to-date the full toolchain is only available for a single Use Case. Future work includes the testing of the developed tools with the other Use Cases, as well as setting up more complex and realistic Use Cases.

5.2 Conclusion

This paper presents Modelica code generation for Building Performance Simulation based on Building Information Models. The focus is on an open-source Python framework to connect BIM with Modelica. The work is embedded in IEA-EBC Project “Computational Tools for Building and Community Energy Systems”, also known as Annex 60. As Annex 60 is an international project with many participants, our approach is an open, adaptable and integrated toolchain with several standalone usable tools. We developed the toolchain to

be generic and applicable for arbitrary Modelica BPS libraries. In this paper we present a Python framework that basically covers three tasks in the overall toolchain: Process control, information pre-processing and Modelica code generation itself. The framework offers the possibility to access the BIM and control the process from a GUI, command line interface or with use of Python scripts. Covered process control includes the choice of a specific file and applying mapping rules or pre-processing steps for different libraries. The pre-processing includes calculation of library and model specific parameters and creation of graphical annotations. To print out the desired Modelica code, we use a template approach. Templates are easy to use and manipulate for each users needs, without necessarily diving into the code itself. We tested the whole toolchain using a first simple Use Case. Future work will include testing the developed process on more detailed Use Cases. The intent of the project is that all developed tools will be available under an open-source license.

Acknowledgement

This work emerged from the Annex 60 project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme. Annex 60 will develop and demonstrate new generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and BIM standards.

We gratefully acknowledge financial support by BMWi (German Federal Ministry of Economic Affairs and Energy), promotional references 03ET1177A and 03ET1177D.

References

- Ruben Baetens, Roel De Coninck, Filip Jorissen, Damien Picard, Lieve Helsen, and Dirk Saelens. OpenIDEAS - an open framework for integrated district energy assessments. In *Proceedings of the 14th IBPSA Conference*, 2015. (submitted).
- Michael Bayer. Mako Templates for Python. <http://www.makotemplates.org/>, 2014. Accessed: 2015-05-13.
- Vladimir Bazjanac, Tobias Maile, James O'Donnell, Cody Rose, and Natasa Mrazovic. Data environments and processing in semi-automated simulation with EnergyPlus. In *CIB W078-W102: 28th International Conference. CIB, Sophia Antipolis, France*, 2011.
- Jun Cao, Tobias Maile, James O'Donnell, Reinhard Wimmer, and Christoph van Treeck. Model transformation from SimModel to Modelica for building energy performance simulation. In *Proceedings of the 5th German-Austrian IBPSA Conference*, pages 242–249, 2014.
- Jun Cao, Reinhard Wimmer, Matthis Thorade, Tobias Maile, James O'Donnell, Jörg Rädler, Jérôme Frisch, and Christoph van Treeck. A flexible model transformation to link BIM with different Modelica libraries for building energy performance simulation. In *Proceedings of the 14th IBPSA Conference*, 2015. (submitted).
- DIN German Institute for Standardization. Radiators and convectors - part 1: Technical specifications and requirements, 2015. 442 - 1.
- Charles Eastman, Paul Teicholz, Rafael Sacks, and Kathleen Liston. *BIM handbook : a guide to building information modeling for owners, managers, designers, engineers and contractors*. Wiley, Hoboken, NJ, 2008. doi:10.1002/9780470261309.
- Energy in Buildings and Communities Programme (EBC). IEA EBC Homepage. <http://iea-ebc.org/>. Accessed: 2015-05-13.
- Marcus Fuchs, Ana Constantin, Moritz Lauster, Peter Remmen, Rita Streblov, and Dirk Müller. Structuring the building performance Modelica model library AixLib for open collaborative development. In *Proceedings of the 14th IBPSA Conference*, 2015. (submitted).
- German Association of Engineers. Calculation of transient thermal response of rooms and buildings - modelling of rooms: VDI 6007-1, 2012. 91.120.10, 91.140.10, 6007-1.
- International Organization for Standardization. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries, 2013. ISO 16739:2013.
- Christoph Nytsch-Geusen, Jörg Huber, Manuel Ljubijankic, and Jörg Rädler. Modelica BuildingSystems – eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme. *Bauphysik*, 35(1):21–29, 2013. doi:10.1002/bapi.201310045.
- James O'Donnell, Richard See, Cody Rose, Tobias Maile, Vladimir Bazjanac, and Philip Haves. SimModel: A domain data model for whole building energy simulation. In *Proceedings of the 12th IBPSA Conference*, pages 382–389, 2011. URL <http://eetd.lbl.gov/node/51892>.
- Qt. Qt Cross-platform application and UI development framework. <http://www.qt.io/>, 2015. Accessed: 2015-05-13.
- Peter Remmen, Jun Cao, Sebastian Ebertshäuser, Jérôme Frisch, Moritz Lauster, Tobias Maile, James O'Donnell, Sergio Pinheiro, Jörg Rädler, Rita Streblov, Matthis Thorade, Reinhard Wimmer, Dirk Müller, Christoph Nytsch-Geusen, and Christoph van Treeck. An open framework for integrated BIM-based building performance simulation using Modelica. In *Proceedings of the 14th IBPSA Conference*, 2015. (submitted).
- Armin Ronacher. Jinja2 Templates for Python. <http://jinja.pocoo.org/>, 2014. Accessed: 2015-05-13.
- Cody M. Rose and Vladimir Bazjanac. An algorithm to generate space boundaries for building energy simulation. *Engineering with Computers*, 31(2):271–280, 2015. doi:10.1007/s00366-013-0347-5.

Michael Wetter and Christoph van Treeck. IEA Annex 60. <http://www.iea-annex60.org/>, 2012. Accessed: 2015-05-13.

Michael Wetter, Wangda Zuo, Thierry Stephane Nouidui, and Xiufeng Pang. Modelica Buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014. doi:10.1080/19401493.2013.765506.

Reinhard Wimmer, Tobias Maile, James O’Donnell, Jun Cao, and Christoph van Treeck. Data-requirements specification to support BIM-based HVAC-definitions in Modelica. In *Proceedings of the 5th German-Austrian IBPSA Conference*, pages 99–107, 2014.

Reinhard Wimmer, Jun Cao, Peter Remmen, Tobias Maile, James O’Donnell, Jérôme Frisch, Rita Streblov, Dirk Müller, and Christoph van Treeck. Implementation of advanced BIM-based mapping rules for automated conversion to Modelica. In *Proceedings of the 14th IBPSA Conference*, 2015. (submitted).

Lessons learnt from network modelling in a low heat density district heating system

Itzal del Hoyo Arce Susana López Perez Saioa Herrero López Iván Mesonero Dávila

IK4-TEKNIKER

Parke Teknologikoa, Iñaki Goenaga 5, 20600 Eibar (Spain)

[itzal.delhoyo, susana.lopez, saioa.herrero, ivan.mesonero}@tekniker.es](mailto:{itzal.delhoyo, susana.lopez, saioa.herrero, ivan.mesonero}@tekniker.es)

Abstract

This paper presents the lessons learnt during the development of a library for the modelling of district heating systems (DH systems), especially focusing on the distribution network. The library was built based on elements from the Modelica Standard Library (Modelica Association, 2012) and the NewThermal library (Lopez, del Hoyo, 2014).

The modelling strategy chosen is described. Furthermore, the requirements established by the DH networks are set out as well as the models created in response to these demands.

Finally, the artificial diffusion phenomenon, present in the simulation of this kind of thermo-fluid systems, is described.

Keywords: district heating modelling, plastic pipe model, buried pipe model, numerical diffusion, courant number, transport delay, thermo-fluid simulation

1 Introduction

District heating (DH) systems produce hot water (or steam) at a central plant. The hot water is then transported through pipes placed underground to individual buildings for space heating or domestic hot water (DHW) generation. Therefore, dwellings in a DH system do not need their own boiler or any other generation system. Hence, a DH system is composed of a generation system, distribution system and consumption side.

The main objective of modelling DH networks is to simulate the rate of energy transport through the system. This transport depends on the water flow through the system as well as on the temperature levels in the DH network. Therefore, as well as in the case of the modelling of other large scale thermo-fluid systems, in the modelling of a DH system there are also fundamentally two different dynamics to take into consideration, flow and temperature dynamics (Frederiksen, Werner, 2013). The most important difference between them is that while changes in the flow are quickly transferred to the whole network, usually in a matter of seconds, temperature changes are transferred relatively slowly, in some cases taking several hours.

The temperature dynamics are therefore the main dynamics in a DH system and it is essential to consider them. In the case of the hydraulic dynamics, the debate is not so clear. The majority of authors work with pseudo-dynamic models, in which the flow and the pressure are calculated based on a static flow model, because in most cases hydraulic dynamics are presumably irrelevant. However, the advantages of a variable flow in a DH system are the low return temperature and the low heat losses in the network. The disadvantage, however, is the risk of insufficient hydraulic balance (Boysen *et al*, 2007). In this case, the dynamic hydraulic balance is key to the automatic control of the flow, so if the systems require a dynamic balance, the hydraulic dynamics of the system have to be taken into account.

In the framework of FP7 European project AMBASSADOR (Autonomous Management System Developed for Building and District Levels) led by Schneider Electric (Onillon, 2014), the dynamic model library for the modelling of DH systems was carried out under Modelica®. The library is composed of models for the generation system (such as boilers or solar thermal collectors) and distribution system (such as pipes, fluid or hydraulic balance valves). The objective of the AMBASSADOR project is the development of and experimentation with systems and tools that aim to optimize the energy usage within a district by managing energy flows, and predicting and mastering energy consumption and energy production. In the case of District Heating systems, control design requires knowing in detail the physical behaviour of the system to be controlled. A library containing detailed models of the components present in a DH system was developed in consequence, including the DH network system on which this paper is focused.

Within this frame, a fully dynamic modelling has been chosen for the modelling of the distribution network, where both the temperature and the flow are simulated dynamically. The modelling is based on Modelica® and uses a Dymola® environment due to the advantages offered (Basciotti, 2012), such as the possibility to implement customised control strategies or the possibility to consider stationary effects and dynamic hydraulic phenomena.

The lessons learnt during the modelling of a low heat density DH network and some of the models developed, as well as the limitations detected, are presented below.

2 Modelling of DH networks

The main components in a DH network, that is, the distribution system, are the medium that contains the energy to be distributed and the pipes through which the water flows through the network. The models described below are focused on these main components, although during the AMBASSADOR project other models, such as the hydraulic balance valve or DH substations, also present in the network, were developed.

2.1 Medium: AdaptableSimpleWater model

The Modelica Standard Library contains different Water models depending on the characteristics taken into account: compressibility, properties variation with temperature, etc. Starting from the most basic `ConstantPropertyLiquidWater` model (which considers the liquid incompressible and its properties constant), to a more sophisticated set of `WaterIF97` models (which take into account compressible fluid with properties dependent on temperature).

The more real approximation implies using the type of water that takes into account both the compressibility and the properties dependent on temperature. However, this kind of simulations have a high computation time and are usually difficult to initialize.

Fluid operation range in DH systems is usually around 30°C-110°C (Frederiksen, Werner, 2013) so if a `ConstantPropertyLiquidWater` model is used significant errors in pressure loss calculation and thus in mass flow rate values could appear. Since although water is essentially incompressible, especially under normal conditions, mass flow rate calculation usually implies the value of fluid viscosity, which depends on temperature. Therefore, a certain error will be involved if constant thermal properties are considered in the fluid model.

With the aim of quantifying the error made, three different water models have been studied:

- `RealWater` model (`WaterIF97_ph`): compressible fluid with properties dependent on temperature. Available in the Modelica Standard Library.
- `SimpleWater` model (`ConstantPropertyLiquid`): incompressible fluid with constant properties established at 20°C. Available in the Modelica Standard Library.
- `SimpleWater70C` model (`ConstantPropertyLiquid70C`): incompressible fluid with constant properties established at 70°C, which is the operation temperature midrange of the DH network modelled. Created for the study.

The comparison has been carried out for two kinds of pipes involved in the DH network,

- A flexible plastic pipe for low-temperature application (DN50) present in the transmission piping
- An usual copper pipe (28mm external diameter) covered by a specific insulation, present in the DH branches

For both analysis, a 10m pipe has been considered and two different mass flow rate values have been used, specifically, the extreme values of the real system (0.016kg/s and 0.25kg/s in the case of copper pipe, and 0.25kg/s and 2kg/s for the flexible pipe). In addition:

- No heat transfer with the ambient has been considered, just 10°C were imposed in the most external layer of the pipes.
- An inlet temperature ramp between 80°C and 60°C (in 2000s) has been simulated.

The pressure drop of both pipes were analysed during the experiments. For the calculation, the “DetailedPipeFlow” option has been chosen for the `FlowModel` replaceable model, in this model, the wall friction in laminar and turbulent regimes is considered.

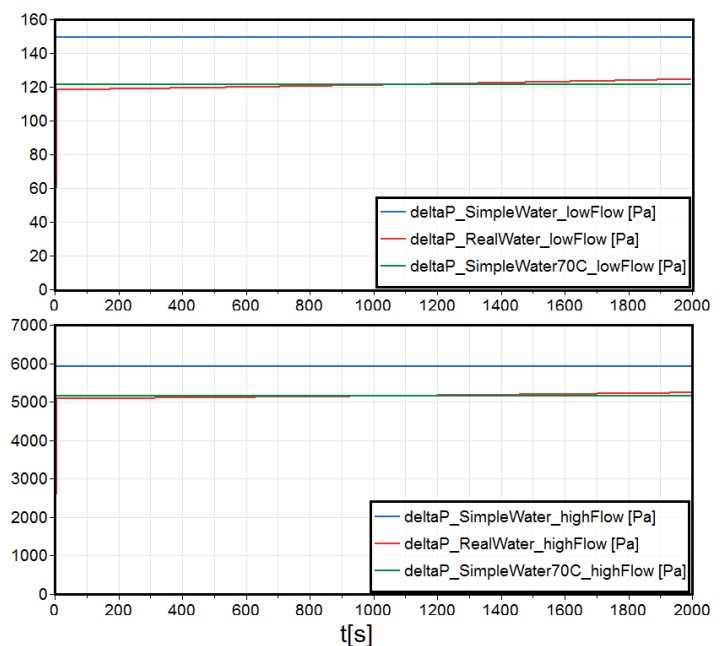


Figure 1. Pressure drop [Pa] in copper pipe (top: 0.016kg/s, bottom: 0.25kg/s)

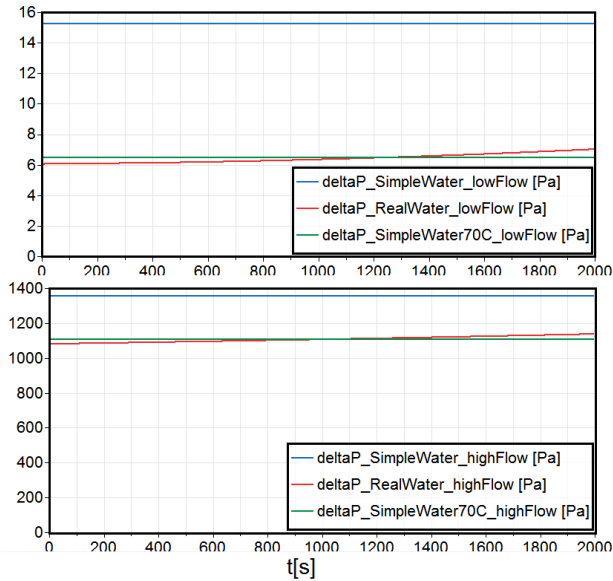


Figure 2. Pressure drop [Pa] in flexible plastic pipe (top: 0.25kg/s, bottom: 2kg/s)

In Figure 1 and Figure 2 the pressure drop values obtained in the simulations are shown. As can be observed, the pressure loss value obtained with SimpleWater70C (in green) is closer to the RealWater mean value (in red) than the SimpleWater (in blue), thus, giving a more realistic behaviour. In addition, the model with RealWater, is 10 times slower than the others.

As expected, RealWater is the only one that shows the pressure loss variation with temperature, as the properties of the other two fluids do not depend on temperature. However, it is observed that the variation with respect to the mean value is very low, so a constant temperature fluid approximation does not seem to introduce a relevant error for the cases, such as the DH system at hand, where the operating temperature range is not very wide.

The comparison has been done considering the RealWater model's results as those that are nearer to the real system's results.

Table 1. Maximum relative error in pressure drop compared with RealWater in copper and flexible pipes

Maximum error [%]	Flexible pipe		Copper pipe	
	Low Flow	High Flow	Low Flow	High Flow
SimpleWater	26.44	16.6	51.9	25.19
SimpleWater70C	2.54	1.4	6.77	2.38

Table 1 shows the high error made with the SimpleWater model in the calculation of pressure drop in comparison with the RealWater results. In the case of copper pipe, in addition, the error made is higher because the flow is on all occasion laminar or near laminar.

Besides, additional simulations have been run for RealWater and SimpleWater70C, but this time the pipe outlet temperature differences have been compared. Once again, simulations have been run for both the copper and the flexible plastic pipes, forcing a transient phase and imposing $T=10^{\circ}\text{C}$ in the last layer of the pipes.

In the case of the flexible pipe, present in the transmission piping of the DH network and therefore more suitable to major changes in mass flow and different levels of temperature, a change in flow (from 0.03kg/s to 2kg/s) has been imposed and different input temperatures have been tested for a 200m pipe. Three inlet temperatures were considered:

- InletTemperature1 = 70°C
- InletTemperature2 = 85°C
- InletTemperature3 = 95°C

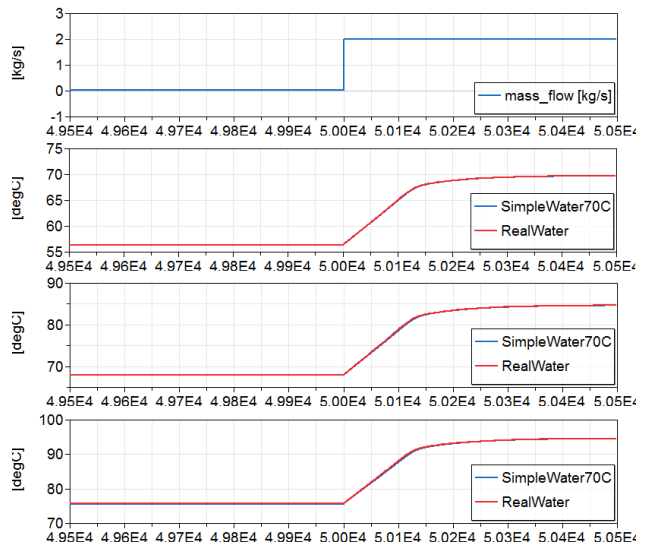


Figure 3. Output temperature differences in a 200m plastic pipe, under a mass flow rate step and coming from three different input temperatures

Figure 3 shows little differences in the output temperature in all the experiments.

Table 2. Outlet temperature comparison. Errors made by SimpleWater70C model compared with RealWater model's results

Inlet temperature [°C]	Abs error [°C]	Mean temperature [°C]	Rel. Error [%]
70	0.18	63	0.3
85	0.334	76	0.44
95	0.474	85	0.56

In the case of copper pipe, a constant mass flow has been considered and a temperature step (from 70°C to 75°C) has been imposed in the inlet temperature.

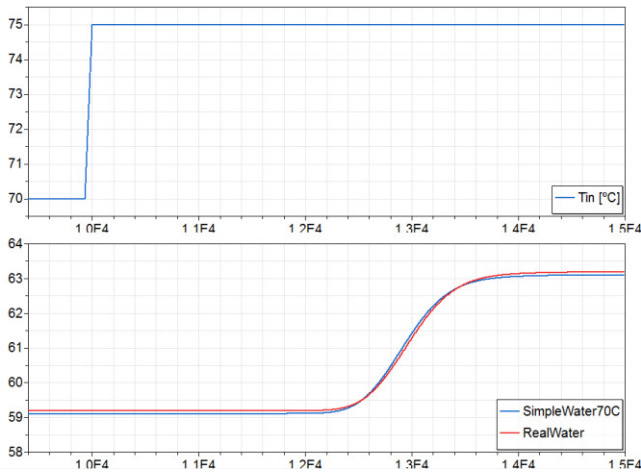


Figure 4. Output temperature in an 80m copper pipe under inlet temperature step and constant mass flow rate (Real Water vs. SimpleWater70C)

In this case, the maximum absolute error is 0.217°C , which, compared to the mean temperature value (72.5°C), results in a maximum relative error of 0.3%.

A similar analysis has been carried out for different applications, such as a little flat plate solar collector plant for generating domestic hot water (DHW), with the same conclusion: it is considered acceptable to use an incompressible and temperature independent fluid, with properties established at operation temperature midrange.

Although the use of a fluid with properties established at operation temperature midrange is not always applicable, in many cases, as above explained, it is a useful simplification. Nevertheless, the calculation of the maximum error made with this simplification is, on all occasions, crucial. Consequently, a new water model has been developed, `adaptableSimpleWater` model. This water model considers water incompressible and that it has properties of the fluid independent of temperature. However, the `adaptableSimpleWater` model calls for the midrange operation temperature of the system to be simulated, and during the whole simulation, it uses the properties of the fluid relative to this established temperature.

Hence, the model has the properties of the fluid declared as a function of the midrange operation temperature. This temperature is established when the `adaptableSimpleWater` class is instantiated. Both, the properties of the fluid and the midrange operation temperature, are declared as a constant.

2.2 Distribution pipes: InsulatedPipe model

The level of detail concerning temperature dynamics, especially the inclusion of the heat capacity of the insulation material and soil, is often omitted in simulations of large scale district heating systems.

These systems also have considerable heat demand and high flow rates resulting in reasonably stable temperature levels within the distribution network. In local small scale and low heat density systems (<0.5 MWh/m), the behaviour of a single consumer is more important thus making a detailed modelling, with smaller time steps being more relevant. This need is further increased in a hybrid system with alternative sources of heat at different temperatures and that are intermittently available, e.g. solar heat. Systems with low heat demand experience significant fluctuations in temperature especially within the connection pipes, i.e. pipes connecting consumers to the distribution network.

While a less detailed model can give adequately accurate results for low heat demand systems on a yearly level, e.g. for heat losses, they are less useful in testing control systems in different use cases and can lead to systems that do not operate as they were designed to according to the simulations.

During the AMBASSADOR project, the dynamic modelling of a low heat density DH network was carried out with control design purpose, therefore, detailed models of distribution pipes were developed considering the heat capacity of all materials present in the pipe. Different versions of the `insulatedPipe` model (Lopez, del Hoyo, 2014) were used as a basis.

The `insulatedPipe` basic model describes the hydraulic and thermal behaviour of any pipe with one or more solid layer(s) assuming radial symmetry in both phenomena. The DH network in this case, however, requires new features in the `insulatedPipe` model, so the basic model has been expanded creating new versions.

2.2.1 Neglecting axial heat transmission

It is common to neglect the axial heat transmission throughout the length of the cover in the district heating and cooling systems pipes (DHC systems). It is done in the most used methods for the modelling of DHC systems, the node method and element method (Pálsson, 2000). This happens because considering the slow temperature dynamics and the poor conductor plastic materials in DHC pipes, the heat transfer in the solid materials of the pipe principally occurs in the radial direction. Traditionally, in the case of modelling DHC systems, there are no major differences between the results of taking into account the axial heat transfer and not taking it into account, but there is a big difference regarding simulation time. Not considering the heat transfer in axial direction reduces considerably computational weight of the simulation, a huge advantage in the simulation of large-scale thermo-fluid systems, because it can significantly speed up simulations.

Hence, based on state of the art, an improved `InsulatedPipe` model was created, called the `InsulatedPipeOptionalAxial` model. This

new model has the same characteristics as the `InsulatedPipe` basic model, but it has the option of choosing between taking into account the heat transfer in axial direction or not considering it. Thus, it is up to the user to decide regarding the aforementioned heat transfer. When the axial heat transfer is neglected the boolean `axialHeat` has to be switched to false.

The experiments done show practically identical results. During the experiment, the pipes were considered adiabatic and they were fed by a pump with a constant mass flow (2kg/s) and a ramp in temperature (from 20°C to 80°C, in 100s). In the Figure 6, the temperatures of three nodes (the first node, the middle node and the last node) are shown for the cases where axial heat conduction is considered and neglected under the same conditions.

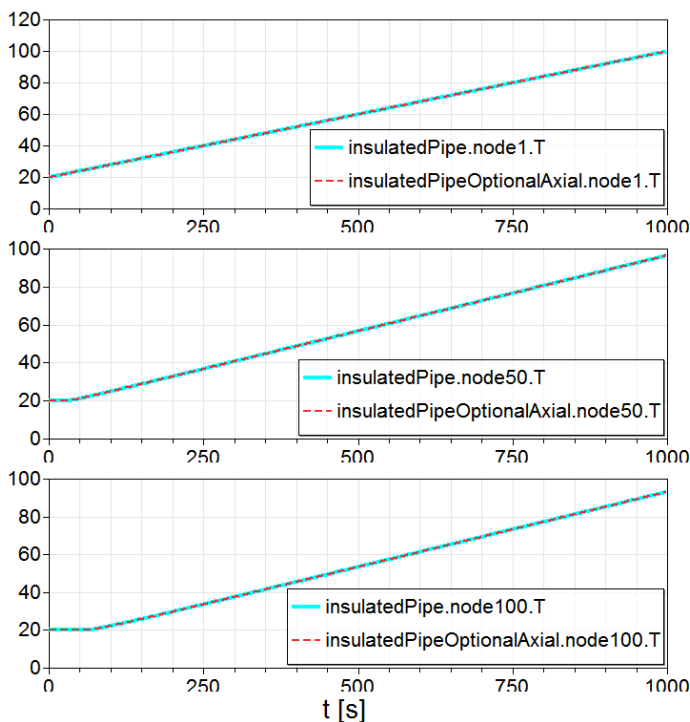


Figure 5. Temperature of three nodes inside the `InsulatedPipe` and `InsulatedPipeOptionalAxial` models

The CPU time consumed, nevertheless, varies significantly, since neglecting the axial heat conduction speeds up the simulation by a factor of 2.69 (the simulation neglecting axial heat conduction can be up to 60% faster).

2.2.2 Buried pipe

The pipes used for transmission in DH systems are usually underground. Buried systems are highly influenced by the soil around them, both the thermal conductivity of the ground and the depth at which they are buried affects the heat transfer in the system, particularly when the insulation has low thermal

resistance (McCauley, 2000). Moreover, soil thermal conductivity changes significantly with moisture content, from 0.14 W/mK in dry soil conditions to 2.16 W/mK in wet soil conditions (Bottorf, 1951).

The most important factors affecting heat transfer are the difference between earth and fluid temperatures and the thermal insulation. Other factors that affect heat transfer are (ASHRAE handbook, 2008):

1. Depth of burial, related to the earth temperature and soil thermal resistance
2. Soil conductivity, related to soil moisture content and density
3. Distance between adjacent pipes.

The mathematical model of DHC system pipes must compute transient heat gains or losses in the underground piping system, and for this, the resistance of the ground has to be considered. The most usual physical model defines thermal resistances between the different materials of the pipe and surrounding ground. That is, in the modelling of a buried pipe, it is divided into three main parts, the surrounding ground, the insulation layers and the water mass.

The surrounding ground is considered, as an infinite inertia. That is, it is supposed that the heat from the buried pipes is not enough to change the temperature and thermal properties of the surrounding soil. Therefore, the influence of the surrounding soil is taken into consideration through a thermal resistance (R_g), depending on the depth at which the pipe is buried (s_d), the conductivity of the ground (K_g) and the external diameter of the buried pipe (D_m), as is shown in the following equations (Pálsson, Larsen, et al, 1999):

$$R_g = \frac{1}{2\pi K_g} \ln\left(\frac{4H}{D_m}\right) \quad (1)$$

$$H = s_d + 0.0685K_g \quad (2)$$

A second thermal resistance (R_H) is presented in the pipe model to take into account the effect of having two pipes side by side in the ground, depending on the depth at which the pipes are buried (s_d), the conductivity of the ground (K_g) and the distance between the centre of the two buried pipes (s_c). Assuming identical supply and return pipes, this resistance is given as (Pálsson, Larsen, et al, 1999):

$$R_H = \frac{1}{2\pi K_g} * \ln\left(1 + \left(\frac{2H}{s_c}\right)^2\right) \quad (3)$$

The buried model has been used for the modelling of a CALPEX® district heating pipe and the validation has been done with the data provided by the manufacturer (CALPEX® technical sheet). The sheet shows two CALPEX®UNO 63/126 buried at 0.60m at a distance

of 0.1m. The ground temperature and soil conductivity are also known:

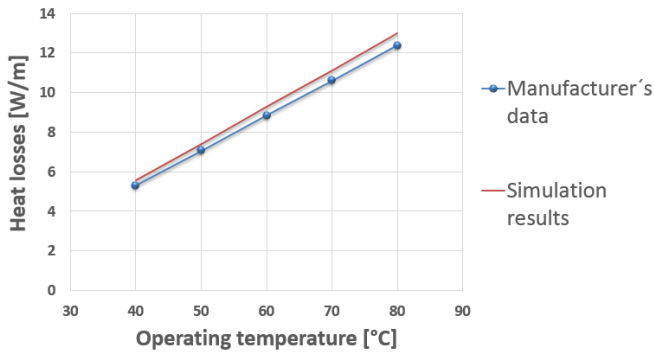


Figure 7. Validation of insulated and buried pipe model against data provided by the manufacturer

The error of the simulation results compared with those provided by the manufacturer is 5.01% at highest water temperature (distribution temperature) and is below the latter value at lower temperatures.

3 Discussion: Artificial diffusion phenomena

During the process of modelling the DH network, a well-known phenomenon has been detected in the simulation results related with the transport delay which arises in any fluid movement through a pipe. The real transport delay is calculated in the following way:

$$t_{delay} = \frac{\rho * A * L}{Q} \quad (4)$$

In the equation (4), ρ is the density of the fluid, A is the transverse area the fluid goes through, L is the length at which the output temperature is observed and Q the flow rate through the system.

In the course of the validation period of the `insulatedPipe` model, the following test was suggested for a flexible plastic pipe of 100 m length, divided into 10 nodes:

- Mass flow: Step signal, the mass flow changes from 1kg/s to 2kg/s at the 30th minute
- Temperature of the incoming water: Double step signal, the incoming water temperature changes from 80°C to 70°C at the 20th minute and again to 60°C at the 40th minute
- It is assumed the ground temperature remains constant (10°C) during the simulation

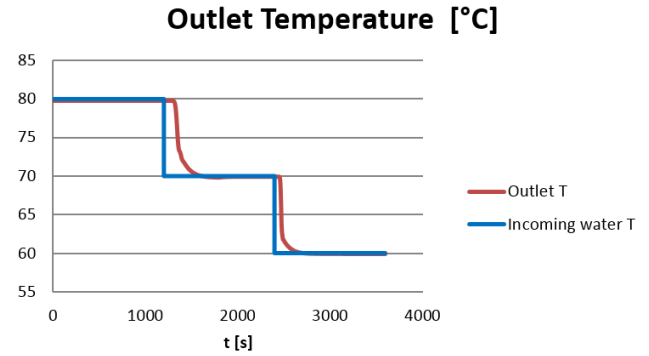


Figure 8. Outlet temperature in a 100m pipe divided into 10 nodes

The real transport delays, calculated by equation (4), that must appear on the outlet temperature behaviour corresponding to the first temperature step and the second temperature step are 130.5s and 65.3s respectively. However, the transport delays in the dynamic response of the model are, rounding, 60s after the first step and 35s in the second one. This phenomenon is known as artificial diffusion.

Artificial diffusion is the consequence of numerical diffusion in the simulation of a continuum phenomenon such as fluid movement, in consequence, the simulated medium exhibits a higher diffusivity than the true medium (Leveque, 2007). In this case, the diffusivity is known as the property of a substance indicative of the rate at which a thermal disturbance, such as a drop in temperature, will be transmitted through the substance. Therefore, the model shows a higher diffusivity than the real case, that is, the drop in temperature through the substance is transmitted faster than in the real case.

The numerical diffusion is often analysed taking the Courant number into account:

$$Cou = \frac{\Delta t * u}{\Delta x} \quad (5)$$

When the Courant number approaches zero the model shows excessive artificial diffusion while $Cou=1$ cases give the exact result.

The Courant number depends on the time step, that is, the time between the current and the previous step (Δt), the element length (Δx) and the flow velocity (u). The latter is imposed by the real case, and in the case of using a variable time-step solver, in addition, in the vast majority of cases, the software itself decides the time-step value (Δt). The integration step size in variable time-step solvers, is chosen in such a way, that the local error is smaller than the desired maximum local error, defined via the relative and absolute tolerances. In other words, a variable (or adaptive) step size implies that the algorithm adapts the step size to meet a local error criterion based on the tolerance (Dassault Systèmes AB. Dymola User Manual, 2015).

Therefore, the user can specify just the element length; accordingly, the artificial diffusion could be improved discretizing more the pipe element. Therefore, in the case at hand, the higher the number of nodes in the pipe the better the results of the model. Considering three different degrees of discretization:

- Considering the pipe divided into one node: $\Delta x=100\text{m}$
- Considering the pipe divided into 10 nodes: $\Delta x=10\text{m}$
- Considering the pipe divided into 100 nodes: $\Delta x=1\text{m}$

The simulation has been carried out with the same tolerance and the exercise's results are shown in the following figure:

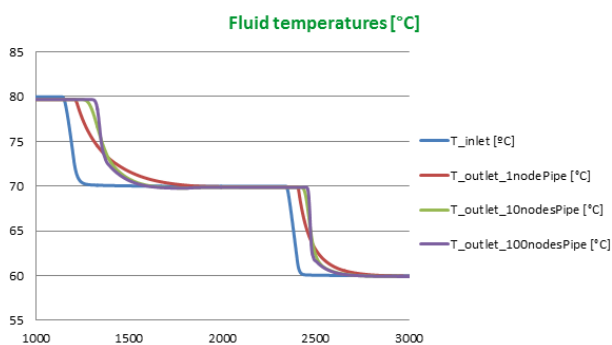


Figure 9. Outlet water temperature for the cases of the pipe discretized into one node, into 10 nodes and into 100 nodes

Figure 9 demonstrates that the fewer the nodes in the pipe the higher the artificial diffusion.

Table 3. Courant number for each case

Mass flow 1kg/s	1 node $\Delta x=100\text{m}$	10 nodes $\Delta x=10\text{m}$	100 nodes $\Delta x=1\text{m}$
Delay in the model [s]	35	65	112
Relative error in the time delay [%]	73	50	14
Cou number	0.007	0.068	0.75
Mass flow 2kg/s	1 node $\Delta x=100\text{m}$	10 nodes $\Delta x=10\text{m}$	100 nodes $\Delta x=1\text{m}$
Cou number	0.015	0.138	1.517

In Table 3 can be seen that the Courant number changes with the number of nodes in the pipe, but as is known intuitively with the case of the 100 nodes pipe (last column), the adequate number of nodes for a specific operating flow rate may not be the best one for other operating flow rate. Hence, the degree of discretization must be in accordance with the operating flow rate range. In addition, when the discretizing level increases the tolerance of the integration has to change accordingly, otherwise, during the calculation a

numerical error may appear which is reflected in the results as artificial diffusion (that is, a thermal disturbance is transmitted through the substance faster than in the real case).

The major disadvantage of this procedure to get the best discretization level of a big thermo-hydraulic system in order to reduce the relevance of the artificial diffusion, is the significant time needed. Since according to the technical support service, using a variable time-step solver is not possible to examine the time step during the course of the simulation. Therefore, the methodology requires running the simulation to have available the necessary data for the calculation of the Courant number. This kind of thermo-fluid simulation, moreover, usually has a high computational weight.

4 Conclusion

The models of the two main components present in a DH network, fluid and buried pipes, have been developed and successfully validated. These models have been developed with elements from both the Modelica Standard Library and NewThermal library (which is in turn based on models from the Modelica Standard Library), and based on the requirements of district heating systems.

Furthermore, the artificial diffusion phenomenon detected in the network models is explained and it is suggested to identify and control its influence through the calculation of the Courant number. The analysis of the Courant number, however, now implies a post-processing of the results since it is not possible to evaluate the time step during the course of the simulation using a variable time-step solver.

Acknowledgements

The research leading to the results presented in this paper (models development, analysis, validation, etc.) has been developed in the framework of project AMBASSADOR, which has received funding from the European Union Seventh Framework Programme [FP7/2007-2013] under grant agreement n°314175

References

- ASHRAE Handbook: Heating, Ventilating & Air-Conditioning Systems, chapter 11. 2008
- Basciotti D., Pol O., A theoretical study of the impact of using small scale thermo chemical storage units in district heating networks. Energy Department AIT, Austria (2012)
- Bottorf, J.D. Summary of thermal conductivity as a function of moisture. 1951
- Boysen, Thorsen. Technical paper, Hydraulic balance in a district heating system (2007)
- Dassault Systèmes AB. Dymola User Manual (2015)
- Svend Frederiksen, Seven Werner. District Heating and Cooling. Studentlitteratur first edition (2013)

- R.J. Leveque (2007). Finite Difference Methods for Ordinary and Partial Differential Equations. ISBN 978-0-898716-29-0
- Susana López, Itzal del Hoyo. Proposal for standardization of Heat Transfer Modelling in NewThermal Library. *10th International Modelica Conference (Lund)*, 2014.
- James McCauley. Steam distribution system deskbook (ISBN 0-88173-303-2), 2000.
- Modelica Association, (2012). A Unified Object-Oriented Language for Physical System Modeling, Modelica®
- Emmanuel Onillon, District energy flow optimization taking into account building flexibilities. International Conference Sustainable Places (2014)
- Halldór Pálsson. Methods for Planning and Operating Decentralized Combined Heat and Power Plants. Riso National Laboratory, Roskilde, Technical University of Denmark, 2000
- Halldór Pálsson, Helge V. Larsen, et al. Equivalent models of district heating system, Technical University of Denmark and Risø National Laboratory, 1999.

Modelica based Design and Optimisation of Control Systems for Solar Heat Systems and Low Energy Buildings

Stephan Seidel¹ Christoph Clauss¹ Jürgen Haufe¹ Kristin Majetta¹ Torsten Blochwitz²

Edgar Liebold³ Ullrich Hintzen⁴ Volker Klostermann⁵

¹ Fraunhofer IIS EAS, Zeunerstraße 38, D-01069 Dresden, GERMANY

² ITI GmbH, Schweriner Straße 1, D-01067 Dresden, GERMANY

³ NSC GmbH, Äußere Zwickauer Straße 8, D-08064 Zwickau, GERMANY

⁴ FASA AG, Marianne-Brandt-Straße 4, D-09112 Chemnitz, GERMANY

⁵ Provedo GmbH, Schweriner Strasse 1, Mottelerstraße 8, D-04155 Leipzig, GERMANY

{stephan.seidel, christoph.clauss, juergen.haufe, kristin.majetta}@eas.iis.fraunhofer.de

Abstract

The goal of the research project enerMAT is the reduction of energy consumption and CO₂ emissions of buildings. Especially solar heating systems are installed in more and more buildings. This paper introduces a novel approach for simulation and optimisation that aims to improve the performance of building controllers and especially solar heating controllers by simulation and model-in-the-loop tests. A new generation of energy-aware optimised building energy management systems (BEMS) will be discussed and its advantages over the older controllers highlighted. The energy-aware optimisation will be shown on a model-based approach with an overall building system model enabling the assessment of the energy performance for different design and operation alternatives of the building automation system in interaction with the building. This system model will allow a simulation-based, energy-aware, global, dynamic, multi-criterial optimisation of BEMS. In this paper, the idea, the approach, and the actual state of the project research is presented with a focus on solar heating controllers.

Keywords: Building, Energy Management, Solar Heat, Controller

1 Introduction

As mentioned in the public media and many scientific studies the energy demand of buildings is responsible for about 40% of the primary energy consumption (European Commission, 2008). This demand is caused by many energy consuming devices and systems such as lighting, water heating, and con-

sumer electronics. The main consumer however is the heating system. There is a two-step approach to reduce the demand of fossil energy in buildings. The first step is to equip buildings with insulation layers and reduce the amount of energy that is emitted into the environment. This passive measure was and still is the main procedure for energy reduction but yields only a small positive effect on the overall energy balance in case the reduction is compared to the expenditure in manufacturing. Insulation however is a prerequisite for low energy houses as it reduces greatly the required heating power. But as explained in the following insulation should not be the only measure for saving energy. Instead of further improving insulation levels renewable energy sources should be employed to reduce the impact on the environment. This is the second step in which fossil fuel heating systems are replaced with renewable energy sources after insulation has been upgraded. Solar energy plays a major role as it is readily available and easy to harvest. Photovoltaic and solar thermal energy systems are installed in low energy houses in order to assist or replace fossil energy consuming systems. Other renewable energy harvesting systems such as heat pumps are an alternative or a support for solar energy. Heat pumps can harvest thermal energy (ground or air) although depending on the heat source their installation can be expensive. Micro wind turbines can be installed as well but are more difficult than solar systems. Their approval by authorities might require costly surveys, moving parts cause higher servicing costs and the noise might cause resentment with the neighbours. Therefore they are not very common in urban areas. The installation of solar energy systems is, depending on size and system, relatively cheap and requires only the installation area on a roof or other exposed area.

Currently almost every newly built detached house in Germany is equipped with a solar thermal heating system in order to meet requirements introduced with the Energieeinsparverordnung (Energy Savings Regulation) (BmWI, 2014). While most houses feature small solar thermal systems which support the warm water supply a growing percentage of the houses built are energy-plus or solar houses that generate most of the energy required for heating from renewable energy sources (solar and geothermal). These houses are in general very well insulated and rely heavily on an optimised heating system that does provide sufficient heat not only in summer but also in winter when the supply of renewable energy is generally low. Therefore it is crucial for these systems that their hardware and software parameters are adapted to the individual building where they are installed. There is a multitude of parameters (w.r.t. to solar thermal heating systems) including size and direction of collector array, type of collectors, threshold temperatures for switching, buffer size and charging procedure as well as feed line temperatures, occupation profiles and heating times and many more. Many if not all of these parameters are established during the planning and construction phase of the building and they are based mainly on experience, especially parameters in the controller. Once installed these parameters cannot be changed at all or only by experts. Therefore it is vital to establish optimal values during the development procedure by using simulation and optimisation.

A solar thermal heating system typically consists of a collector array, a huge water-filled buffer storing the thermal energy, a circulation pump on the source side and floor heating, hot water heating and circulation pumps on the consumer side. All of these devices are controlled by a dedicated stand-alone controller. Among others the universal controller 1611 (Universalregelung UVR 1611) from the Austrian manufacturer Technische Alternative is a well-

known and widely used device. This device is programmed by using a PC and parameterising and connecting function blocks in the programming tool from a preconfigured library.

This paper will present an approach to use simulation and model-based optimisation to validate the planned solar-heating system and find ideal parameters for installation and control unit.

Furthermore this paper will highlight the use of simulation and optimisation for low energy buildings in general and for controllers for solar thermal heating systems in special. Section 2 will explain the controller in some detail and will also discuss the targets for simulation and the expected benefits. The following section 3 will shed some light on the approach for simulating the solar thermal heating system in conjunction with control systems and the surrounding house. This will be further elaborated in section 4 where two use cases and demonstration projects are presented. Section 5 will give an overview of the current status of the project and an outlook. The paper will conclude with section 6.

2 Targets

The UVR 1611 control unit introduced in section 1 is a typical controller for HVAC systems and was developed by the Austrian company Technische Alternative since 2000. It is tailor-made for HVAC and solar heating systems and features a wide set of functions for this purpose. The controller has 16 sensor inputs (typically temperature sensors), 4 speed outputs (e.g. for circulation pumps), 7 relay outputs (for opener/closer switches or valves), a CAN bus connection and several extension modules to add more relay outputs or LAN connectivity for viewing and operating the controller via internet.

Although described and marketed as freely programmable this should be understood in a different way. Programming languages such as IEC 61131-3's instruction list or ladder logic are not available. The closest equivalent in 61131-3 would be the function block diagram (FBD) although the function blocks in the UVR cannot be connected with each other. A UVR configuration (or sometimes called program) consists of several function blocks that process inputs and flag values and write outputs and flag values. Typical function blocks are provided by a library but it is not possible to write own function blocks. This is in most cases not necessary because typically the UVR is not programmed by an engineer but rather configured by a technician. Functions blocks are ranging from simple logic functions (AND, OR, FlipFlop), compare function, timers,



Figure 1: Universal controller UVR1611

counters and clocks to control functions such as PID control. Many function blocks are designed for typical HVAC control tasks such as heating circuit controller, mixer control, load pump control, solar control, boiler cascade control and many more. This concept is part of the UVR's success (tens of thousands sold units) because typical tasks can be solved by configuring a few function blocks instead of programming everything from scratch. The configuration, which is done on a PC and downloaded onto the device, is fairly easy to do. Function blocks are pulled from the library and the parameters are edited. Many function blocks provide switches for behaviour changing (e.g. P-, PI- or PID controller) and cover a wide range of required functionality. Internet forums are full of example configurations and also the manufacturer provides a range of configuration examples for download.

The UVR's success is due to its versatility and its scalability. Typical use cases are heating systems in detached houses that go beyond the standard natural gas powered heating system which are controlled by the integrated controller of the condensing boiler such as solar thermal heating systems, geothermal heating systems or older systems that require an extension or a retrofit. Within the enerMAT project UVR controllers were installed by the FASA AG as control unit for the heating system of so called solar houses and offices with the trade name ENER-GETIKhaus100. These houses are not standard low energy houses, in fact they have a higher demand in heating energy than typical low energy houses, but the heat is provided entirely by renewable energy sources. Typically the UVR would control the solar heating system, its circulation pump and the corresponding valves, the loading of the buffer with solar heat and the circulation pump of the floor heating. The solar heating system in such houses by FASA has the aim of providing up to 92% of the heat required by the house and inhabitants by means of solar energy. The remainder is covered by a wood-burning stove. Typically one UVR is sufficient to cover all the pending control tasks in a detached house.

Other use cases, especially more sophisticated or larger installations can be resolved by installing several UVR units which communicate by CAN bus with each other. Hence UVR 1611 controllers can be installed wherever an off-the-shelf solution for heating systems is not available or not desired because of compatibility issues between components of different manufactures or missing features. Since the UVR 1611 is an older controller type it is currently phased out and replaced by a newer type, the UVR16X2

which is similar but not identical in form and function to the 1611 model.

3 Approach

As mentioned already in Sections 1 and 2 the lack of a methodology to obtain optimised parameters before the actual implementation on the HVAC controller leads to non-optimal behaviour of the heating system which could result in poor heating performance and energy wasting. Parameters are often estimated or based on empirical values which are not necessarily ideal for the particular heating system. Therefore a simulation-based commissioning and optimisation is required to obtain ideal functionality and parameters for the heating system controller such as the UVR 1611. Simulation or virtual commissioning is almost non-existent in the field of such control units. This is vastly different for industrial programmable logic controllers (PLC) such as PLCs for intralogistic systems (Seidel, 2012) which are tested and optimised on a virtual model of the plant before the real commissioning takes place. This has various advantages such as thoroughly tested and mature software with fewer bugs, shorter commissioning and project time and reduced costs. Unfortunately virtual commissioning doesn't play any significant role in building control systems. One of the enerMat project's aim was to provide engineers with the tools required for a continuous workflow for design, test and optimisation of building controllers and systems of which one subsystem the UVR 1611 is.

Several options for simulation, emulation and virtual commissioning have been analysed. The lack of a software simulator for the UVR 1611 controller prevents any form of software-in-the-loop (SiL) approach. Software available for the UVR is limited to the programming system TAPPS which features no simulation or emulation mode. If such an emulation tool would exist a co-simulation solution would be feasible. Hardware-in-the-loop (HiL) approaches would be feasible by connecting the hardware controller by means of an I/O interface and FMI to the simulation. The big issue however is the missing time synchronisation between controller and simulation tool. The hardware controller would always be restricted to real-time, and simulation runs over a long time span such as a year would be difficult because of the required amount of time. Therefore a model-in-the-loop (MiL) was favoured and found to be the most promising approach which avoids the error-prone coupling of tools or systems. By applying MiL the engineer can concentrate on the simulation tool and the designed controller and building

model. Other advantages are: symbolic preprocessing of the model and thus a more reliable and faster simulation.

To create a model of the UVR controller (and any other controller) a detailed functional description is required. This can be either the program code or a system specification. Since the source code was not available the function block specification (TA, 2014) was used as a basis for the design of the controller model. Such a model consists of two submodels. One is the functional model; the other is the behavioural model (Seidel, 2009). The latter describes how the controller works internally, for instance what amount of time is consumed for calculating new outputs from a change of inputs (cycle time) or in which sequence the user program is executed. Knowledge of this behaviour is important for fast processes such as manufacturing machines (Seidel, 2011) or car engines. In the case of building controls this can be neglected and the designed behaviour model has no cycle time because the controller's cycle time which is in the range of milliseconds is very small compared with time constants of the building and heating system (minutes to hours).

The functional model consists of the user program which is the function block structure with corresponding inputs, outputs and parameters. Therefore to create a model of the user program a library of the available function blocks is required. Components of this library are models of each function block having the same inputs, outputs and parameters as their counterpart in the real-world controller. The specification of each function block was analysed, inputs and outputs connectors were created and the function was modeled by means of algorithms and equations or in case of more complex blocks as a group of interconnected elements from the Modelica Standard Library (MSL). As already mentioned the complexity of the function blocks was very diverse. Simple blocks, such as the comparison block, required just a few lines of code as algorithm section as shown below.

```

when valueA > valueB + diffOn then
  valAGreater := true;
elsewhen valueA < valueB + diffOff then
  valAGreater := false;
end when;
valALower:=not valAGreater;
if enable then
  valAGreaterOut := valAGreater;
  valALowerOut := valALower;
else
  valAGreaterOut:=false;
  valALowerOut:=false;
end if;

```

Modelica Code fragment of comparison block

It is worth noting that many of the function blocks implement margin parameters for signal inputs to filter noise or minor oscillations and to prevent a frequent switching of outputs.

On the other hand, the PID control block was considerably more complex because the function block can be set to three different behaviours (absolute value controller, differential value controller and event triggered controller). The final version of the PID block consists of 11 MSL blocks and several lines of code. A third example is the synchronisation block which generates user defined outputs signals according to different times of day or week. Although this function is not very complex the final version contains 230 lines of code but no MSL blocks.

Testing was very important as testing was an iterative process during the development of the function blocks. In a first step for each block an individual

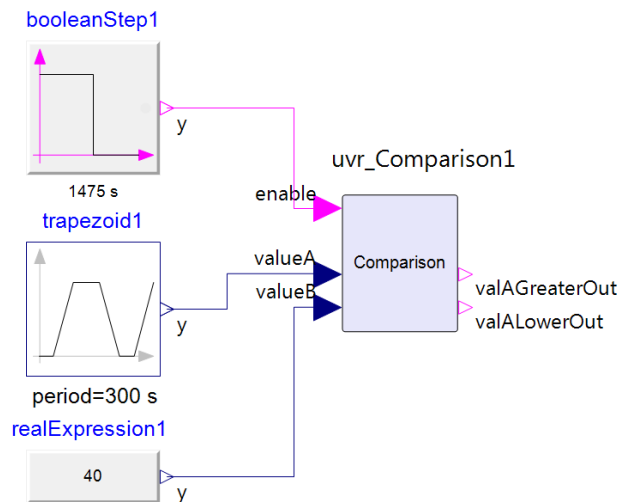


Figure 2: Comparison Function block test model

test model was designed which consisted mainly of signal functions from the MSL to generate input signals and used to test and verify the function blocks (test model for Comparison block is shown in Figure 2). In a second step we generated test signals by using the UVR 1611 simulation board which is an additional hardware module for the UVR and features 16 potentiometers with which temperature inputs can be set manually. After loading user programs consisting of just the function block under test into the UVR the inputs were altered and the corresponding outputs signals were recorded. These test signals were then fed to the function block under test and the recorded outputs were compared to the simulated ones.

As a third step a model of a detached house was created consisting of several parts:

- model of the UVR control program

- model of the solar heating system
- model of the buffer for storing thermal energy
- model of the building including a single room with floor heating.

This model was developed using components from the GreenBuilding Library (Schwan, 2012) which is part of SimulationX. This library provides model components for buildings and renewable energy systems. This model (Figure 3) was kept simple in order to detect errors in the UVR controller model which consists of several different control blocks (solar control, comparison, timer and others). The focus was on the interaction of the UVR control blocks with the rest of the model components and on the ratio of simulation time vs. computing time. The required amount of computing time was relatively small and in a few cases where an increase of computing time was registered, this was due to programming errors or faulty parameters. Both problems caused the generation of too many events mainly because thresholds were too close or states in the function blocks not stable.

Not all available function blocks for the UVR controller were modeled because there were too many of them and some are rarely used. Therefore we created a subset of the 12 most used function blocks which were modeled in Modelica and implemented into a library:

- Solar control
- Start function
- Cooling function
- Comparison
- Load pump
- PID control (speed control)
- Analog function
- Timer

- Time Switch
- Synchronisation
- Heat quantity counter
- Counter

The library of UVR function blocks is thus useable and can be extended to its full extend. However development was halted after around 60% of function blocks had been modelled because the new controller UVR 16X2 which replaces the older 1611 will feature a slightly different set of functions and will not be compatible with the 1611. In addition the main function blocks required in the project for UVR 1611 were modelled.

4 Use Cases and Demonstrators

After the required function blocks were modelled and tested we were able to implement the larger model of a refurbished office building in Chemnitz which is also a demonstrator in the enerMat project. The main building components that were included in the model are:

- Solar heating system with 270m² solar thermal collector area providing up to 80% of the annual heat energy (90% was planned)
- 110m³ heat storage buffer
- Building with 2 floors and 1150m² heated area
- Wood burning stove as backup heating system for cold and darker winter months (providing the remaining 20% heat energy)
- Heat pump to rearrange heat distribution in the buffer which also extends the range of the buffer. It has no external heat source and is used solely with water from the buffer.

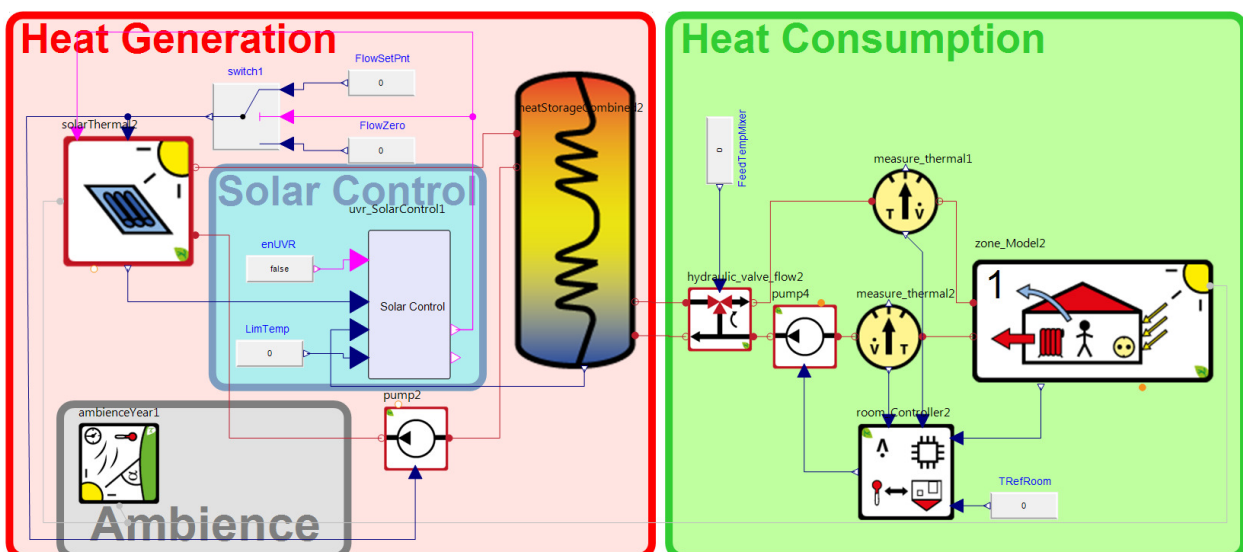


Figure 3: Model of a house with solar heating system and a UVR function block Solar Control

The refurbishment of the office building was planned and designed not as a low energy building but as a building that is sustainable and requires almost no fossil fuel for heating apart from the electrical energy of the heat pump which is active for around 10 days per year during the winter months. Therefore the building lacks typical features of low energy buildings such as air condition, automatic windows, etc. The Modelica model for this building is split into the energy source and energy demand model which were developed in parallel.

4.1 Energy source model

The energy source model contains the solar thermal collector array, the UVR controller, the storage buffer, heat pump and wood burning stove along with several pumps and valves.

Combined with a dummy heat sink model consuming heat from the buffer this model was used to validate the UVR controller model by comparing simulation data with sensor readings obtained from the real controller. Typical signals were temperatures in the collector and buffer, pump and valve signals, power and heat levels. Simulation and measured values were relatively similar with some deviations that can be attributed to the temporal resolution of the meteorological data which was one hour (mean values for one hour: temperature, solar radiation, wind, etc...) and the one-point character of the collector model. The UVR controller model was created by using real UVR controller in the office building as template and was modeled one to one w.r.t. to function blocks, inputs and parameters.

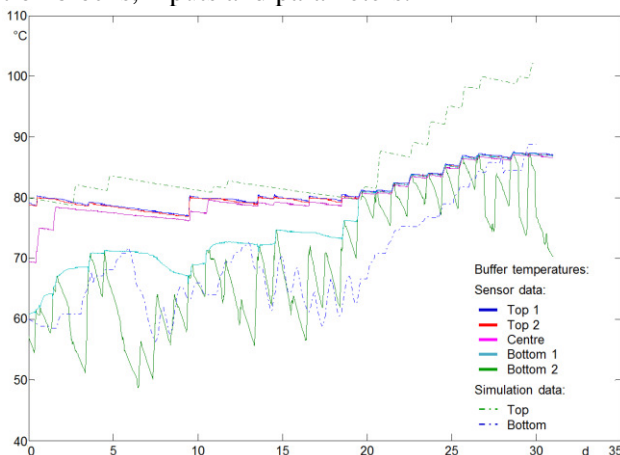


Figure 4: Temperature in 110m³ buffer (30 days)

Figure 4 shows the simulated buffer temperature distribution from the energy source model compared to measurements over a simulation period of 30 days. Input for the model was meteorological data recorded during the same period as the measurements were taken. (Note: Local weather data was not available,

weather data was recorded at a weather station around 60km from building). The general trend between simulation model and measurement data is identical and deviations are due to a very simplistic model for heat consumption which extracts heat differently from the buffer than the real-world heating system. Real world data from the consumption side which would make the model more exact is not available.

4.2 Heat consumption model

Components from the GreenBuilding library were used to model the office building complete with building zones, heating system, inner loads and shading. Several parts of the model had to be created from scratch where the GreenBuilding library did not provide sufficient components. These were models for:

- Heat pump and control
- Wood burning stove and control
- Distribution valve
- Heat meter
- Heating circuit manifold
- Time switches
- Two point temperature controller

and several other purposes. The solar heated office building has 53 rooms on 2 floors. Each room is equipped with a heating circuit which is supplied with warm water from a central heat storage tank. This tank is heated with solar power produced by a solar heating system which is part of the energy source model.

The model of the overall building covers 25 building zones each equipped with a heating system (heat consumer). Several neighbouring office rooms had to be combined into one building zone to speed up simulation and because the maximum amount of building zones in one model was limited. Modelling itself was quite time consuming because every boundary had to be parameterised manually with the corresponding parameters (thickness, thermal transmittance, area, material, ...) taken from Excel sheets. After the completion of the demand part of the model several simulation runs were analysed in order to assess the models conformity to the real-world and numerous test cases were defined such as:

- Test of the heating behaviour in room:
 - maximum heating
 - No heating (neighbouring rooms are heated)
- Room occupation with different number of persons

- Windows opened and closed for certain amount of time
- Lighting (halogen bulbs) in room activated

Sensors at different positions recorded the temperature during the tests and the readings were later compared to the simulation results. Typical deviations initially observed between measured and simulation data were:

- Slope of temperature too steep due to simplified heating model not containing insulation layer between room and underfloor heating
- Simulated temperature 1K too low due to not modelled internal gains and/or misplaced temperature sensors (main logging sensor was attached to ceiling)
- Solar gains because of sunlight shining into the room too low resulting in lower temperatures in simulation
- single point room model too coarse (just single temperature instead of temperature distribution)

It was the aim of these tests and simulations to determine the accuracy of the simulation model. All tests were conducted in one office room because the building was in constant use and tests with the heating system would have caused discomfort for the inhabitants. The aforementioned deviations led to some improvements in the simulation model such as an added first order delay for the heating power. An upgraded heating model would have produced better results but the heating model from the library was protected and could not be extended. Results from the adjusted model are compared to measured data in Figure 5.

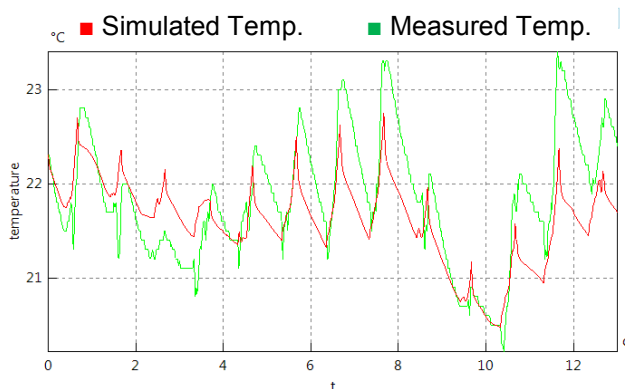


Figure 5: Simulated temperature compared to measured temperature in office room (13 days)

The complete building model with heating system was also simulated for one year and meteorological data for a test reference year (TRY) was used to obtain the energy demand of the building. Results (annual heat consumption in kWh/m²) were then com-

pared with values calculated by the architect for the buildings energy performance certificate and were remarkably close (simulation 57kWh/m²; certificate 52kWh/m²). The 57kWh/m² are provided by solar heat and in case the solar heat buffer is depleted by a wood burning stove or the heat pump.

Simulation was then employed to find answers to the following question. Is it useful to program the heating controller to lower the set point temperature at night and/or at weekends?

Simulation results did not provide a simple answer as shown in Figure 6. The underfloor heating system is characterised by a couple of attributes that make it very slow: two-point controller with an on/off valve; long time constants of up to 12h, low feed temperature (around 32°C but down to 29°C in winter). All these factors contribute to a very slow heating system which makes energy saving measures like lowering the set point temperature difficult. The main issue with a lowered set point (red line) is that reheating the room in the morning or after weekends takes too long and a comfortable room temperature (blue and green lines) is not reached in time as shown in Figure 6.

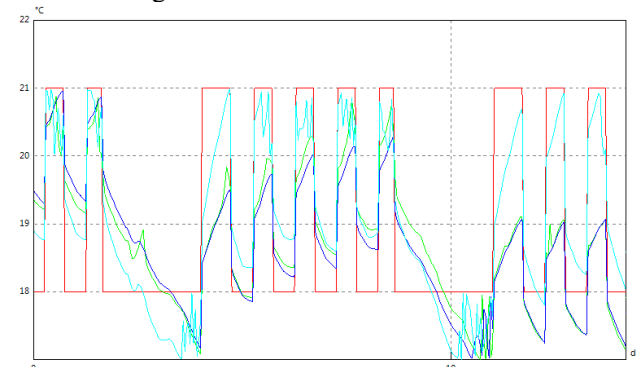


Figure 6: Temperature setpoint lowered to 18°C on weekends and after working hours

This result was an important finding and also a proof that energy saving by lowering the temperature during night and weekends comes at a price of reduced comfort at least in this particular building. Subsequently this also demonstrates the need for an intelligent heating controller algorithm that calculates the required preheating time w.r.t. current room temperature, outside temperature and available heating power (feed temperatures) and next occupation. Such a controller is currently under development and cannot be implemented using the function blocks of the UVR controller as they are not flexible enough. Any form of intelligent algorithm must therefore be implemented on a top level controlling system such as a BEMS or BACS.

4.3 Combined energy and building model

The energy source model and demand model were then combined into one single model. This model was rather large and separate development was crucial for handling and verification of each part. The one year simulation produced results that were already observed by the buildings occupants. Although the solar heat harvested during summer and stored in the 110m³ water buffer will last until December or January depending on weather conditions, it is not sufficient to provide enough heat until spring. It was therefore necessary to heat the buffer by means of a powerful but manually operated wood burning stove. In addition the concept of installing a heat pump to rearrange the temperature distribution in the buffer was simulated and evaluated and subsequently implemented. While this is a reasonable approach and helps to provide the required feed temperatures for the heating system the amount of electrical energy needed by the heat pump cannot be neglected. Therefore several tasks have to be targeted with regard to the control system:

1. The UVR controller has to be as effective as possible, thus harvesting as much solar energy as possible.

The validation of the UVR parameters with the combined model was done along with a manual variation of certain UVR parameters to find better values. Due to the limited accuracy of the collector model this was found to be not promising. Especially time constants could not be validated because of the one-point collector model. Therefore the reaction of the collector model was too fast and for a model-based optimisation of the UVR parameters the model of the collector was inadequate. Development of a more detailed collector model was not part of the project.

2. The heating system should save energy by lowering set points in non-occupied rooms or during nights and weekends. As discussed in section 4.2 a possible solution would be an intelligent controller or BEMS algorithm. An approach to find a solution to this task was developed in this project as well and is discussed in (Majetta, 2015).
3. The heating support from the wood burning stove has to be optimised while the runtime of the heat pump to save electrical energy must be minimised. As a solution for this problem a BEMS algorithm has been developed which constantly monitors the feed temperatures of the heating system from the buffer and informs the occupant when additional heat from the wood burning stove is required. Such a system is currently not installed and the occupant must decide

for himself whether it is time to start the stove or not. The BEMS module informs the inhabitants about the right time to start the stove and will in addition start the heat pump if the stove's heat is not sufficient or if no occupant is around to start the stove (during night or on weekends). The heat pump however is only used for redistribution of the temperature layers in the buffer and takes heat from the colder bottom of the buffer to heat up water at warmer top of the buffer where the feed connector to the heating circuits is installed.

Especially for tasks 2 and 3 simulation-based optimisation was employed but not really achievable due to very long simulation times of the model. Initially it was planned to simulate a whole year with the combined model and use this model also for optimisation. But a year simulation required around 5 hours computing time which is far too long for optimisation purposes. Therefore the model was analysed in order to find the part that requires the most computing time. It was found that the different building zones and their heat exchange through boundaries were responsible for roughly 80% of the required time. Hence a less detailed model was developed with just 2 building zones instead of 26. The 2 zones are representing the ground floor and the first floor. The energy demand of this model was higher due to the size of each floor and the set point temperature of 22°C whereas in the detailed model several rooms had a set point temperature of only 18°C such as storage rooms and libraries. The simplified model was much faster and required around 20 min for a year simulation and was thus fast enough for simulation-based optimisation.

4.4 BEMS development

Development of the BEMS was done in two steps. At first a statechart was created which represents the different states of the buffer and triggers the start of the stove and heat pump depending on certain conditions and parameters. The correct function of the statechart was then tested together with the simplified model. Then the fixed parameters were replaced with a function that evaluates buffer condition and weather forecast. This function takes into account whether warm and sunny weather is expected or cold and overcast weather. Depending on the result the heating command will be given earlier or later. In order to enable optimisation the function was equipped with certain weighting factors and a target function to be minimised.

Optimisation was then employed to find ideal parameters for these factors. In order to make the mod-

el ready for the optimisation framework it had to be exported as an FMU (functional mock-up unit) and was then executed within the MOEA framework (multiobjective evolutionary algorithms).

A PSO (particle swarm optimisation) algorithm (Kennedy, 1995) from the MOEA framework was employed to find optimised parameters for the BEMS function.

	Fixed Param.	Non-Optimised Param. Funct.	Optimised Param.Funct.
Minimised Model			
En. Heat Pump [kWh]	4437	3976	3877
En. Stove [kWh]	14440	16320	17437
Temp.Violation [days]	66	64	62
Full-scale Model			
En. Heat Pump [kWh]	3638	3632	3656
En. Stove [kWh]	13881	15114	16732
Temp.Violation [days]	75	64	52

Table 1: Simulation and Optimisation Results for BEMS controlled stove and heat pump

A typical optimisation run with a 6 months simulation period and the following PSO parameters (6 particles, 50 iterations) results in 300 simulation runs and takes around 23 hours to complete. The FMU of the simplified model was used for the optimisation. Optimisation targets were to minimise the operation time of the heat pump and thus reduce the amount of electrical energy required and to reduce the temperature violation which is triggered if the feed temperature of the heating system is below a certain threshold (30°C). An important restriction for the overall system is that the stove can only be operated between 7am and 5pm on weekdays because it requires manual loading and starting.

The optimisation results in Table 1 show a small reduction in the energy consumption of the heat pump compared to the non-optimised or fixed parameters. In addition number of days where low feed temperatures were registered is also lower which can be seen as a higher level of comfort for the occupants. This results in a higher heat production of the stove which is equivalent to a longer operation time (stove is started on more days in winter). The required firewood was not taken into account as an optimisation target as because it is available for free. An important question was however if the simulation and optimisation results (parameters) of the reduced model can be transferred directly to the full scale model and will lead to improvements as well. With 5 hours computing time the optimisation for this model would have taken around 60 days. The optimised parameters from the reduced model were having the similar effect in the full scale model (see Table 1) however the heat pump energy was not significantly reduced but the temperature violation was lower. It

was found that the limited operation time slot of the stove is the main factor which prevents a further reduction of the heat pump energy.

5 Actual State and Outlook

The UVR library for Modelica has been tested and used in this project but it is not finished as of yet due to the discontinuation of the controller from the manufacturer. The function blocks however are useable not only in UVR related projects but also for other purposes because they implement typical functions that are required in building control systems. It has been found that the design of a model from scratch for a single family house will be too time consuming and the subsequent energy savings will not cover the cost of simulation and optimisation. Therefore it is planned to create models for standard UVR applications and standard houses of one manufacturer. The simulation of these models does not require modelling skills and can be used to test UVR parameters for certain different configurations such as different collector area or buffer size, different location, usage patterns and house parameters. This will enable the HVAC planner or engineer to install the controller with optimised parameters and will subsequently cause a higher performance of the solar heating system combined with an increased convenience for occupants without the need to develop a new model for each project.

BEMS development is currently under way and the simulation results are very promising. It has been shown in this paper that the implementation of a BEMS can solve certain tasks that cannot be easily conquered with a stand-alone controller such as the UVR. While the UVR and similar controllers are great for standard applications more difficult tasks require control systems that go beyond the limits of stand-alone control units. Our BEMS approach can be such a system. Simulation and optimisation will play a significant role in the future. This paper tried to highlight the use of both strategies in the development and validation for building controls.

6 Conclusion

In this paper several ideas w.r.t building control systems, simulation and optimisation were discussed. As of yet control systems in buildings are stand-alone units and energy efficiency cannot be fully achieved with these units because the standard commissioning procedures rely heavily on experience rather than optimised and tested parameters. There-

fore a library of controller function blocks has been developed that enables the engineer to test and optimise a stand-alone building controller in a simulation environment. This library is compatible to the program on the real-world controller so that the parameters established and optimised in simulation can then be transferred to the controller. The limits of this approach were also discussed and a different approach for the implementation of additional functionality into a Building Energy Management System (BEMS) was demonstrated. An optimisation algorithm was implemented to find BEMS parameters that are highly energy efficient and ensure a better comfort for occupants.

Acknowledgement



We would like to express our gratitude to the Austrian company Technische Alternative for giving us the opportunity to use their controller as a basis for our work and for their support in publishing this paper. In addition we would like to thank our student Leonie Sperner who developed an essential part of the UVR Modelica Function Block Library.

References

Bundesministerium für Wirtschaft und Energie *Energieeinsparverordnung* 2014

European Commission: *EU Energy and transport in figures, statistical pocket book* 2007/2008

Kennedy J., E. R.: *Particle swarm optimization*. IEEE International Conference on Neural Networks, (S. 1942 - 1948). Bur. of Labor Stat., Washington, DC, USA, 1995.

Majetta, K., Clauss, C., Haufe, J., Seidel, S., Blochwitz, T., Liebold, E., Hintzen, U., Klostermann, V.: *Design and Optimization of an Energy Manager for an Office Building*, ASIM/GI-Section Workshop – Simulation of Technical Systems & Methods in Modelling and Simulation, Stralsund, June 2015

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, Munich, September 2012.

Seidel, S., Donath, U., Haufe, J.: *Approach to a Simulation-based Verification Environment for Material Handling Systems*, 17th IEEE Conference on Emerging Technologies and Factory Automation, Krakow, September 2012

Seidel, S., Klotz, T., Donath, U., Haufe, J.: *Modelling the Real-Time Behaviour of Machine Controls using UML Statecharts*, 15th IEEE Conference on Emerging Technologies and Factory Automation, Bilbao, September 2010

Seidel, S. Donath, U.: *Error-free Control Programs by means of Graphical Program Design, Simulation-based Verification and Automatic Code Generation*, 8th International Modelica Conference, Dresden, March 2011

Technische Alternative: *UVR 1611 Frei programmierbare Universalregelung Bedienung, Programmierung, Montageanleitung* Ver.4 2014

How to Shape Noise Spectra for Continuous System Simulation

Andreas Klöckner¹ Andreas Knoblach¹ Andreas Heckmann¹

¹DLR German Aerospace Center, Institute of System Dynamics and Control, 82234 Weßling, Germany,
andreas.{kloeckner,knoblach,heckmann}@dlr.de

Abstract

Noise for continuous-time system simulation is relevant for many applications, where time-domain results are required. Simulating such noise raises the need to consistently shape the frequency content of the signal. However, the methods for this task are not obvious and often form filters are approximated by state space implementations. In this paper, we address the problem with a new method relying on directly using the specified power spectral density for a convolution filter. For the example of railway track irregularities, we explain how to derive the required filters, implement them in the open-source `Noise` library, and verify the results. The new method produces correct results, is very simple to use, and enables new features for time simulation of physical systems.

Keywords: Noise, power spectral density, track irregularity

1 Introduction

Modeling stochastic signals is of interest in a wide range of applications, such as sensor modeling, aerodynamic turbulence, and rail irregularities. Previous Modelica libraries, such as the `Statistics` library (Haase et al., 2008), allow to precisely define statistical properties of such signals. However, other properties of the noise signals such as the underlying random number generator or the signal's frequency content could not be modeled as conveniently. A Modelica `Noise` library has thus recently been released in order to enable the engineer to conveniently and consistently define noise signals (Klöckner et al., 2014). It is intended to include a subset of sampled noise generators and standard distributions in the Modelica standard library. The remaining functionality will still be available in the `AdvancedNoise` library.

The `Noise` library also introduces a new class of random number generators: `DIRCS Immediate Random with Continuous Seed` allows to generate random numbers from an input signal without internal states. It thus eliminates the need for time-events, but can be used to generate a random signal directly from the `time` variable. This has been shown to positively af-

fect the simulation performance (van der Linden et al., 2015). Additionally, it allows to define noise signals in dimensions other than the time. This is advantageous in several applications. Rail irregularities e.g. are typically defined with respect to the location on the track. Turbulence models used in aviation also assume a static wind field flown through by the aircraft.

Additionally, the frequency content of noise input to a system must be carefully modeled. It is usually specified by a power spectral density (PSD). If a linear time invariant (LTI) system model is considered, the PSD can be applied in the frequency domain by multiplying the PSD with the squared transfer function of the model. See Frederich (1984) for a railway application and EASA CS-25 (2013) for a typical aircraft application. If a nonlinear model has to be simulated in the time domain, a suitable filter transfer function must be derived from the PSD. In the case, that the PSD is a rational function w.r.t. to the *squared frequency*, a spectral factorization of the PSD can be derived analytically. See Liepmann (1952) for an aeronautical example. Otherwise, the PSD must be approximated by a suitable function. An alternative approach is the recently developed Fractional-Order Modeling Toolbox for Modelica (Pollok et al., 2015), which allows to simulate also non rational transfer functions.

In summary, it is not at all obvious how to parametrize these frequency properties. We thus present a systematic method to shape the frequency content of noise signals. The contributions of this paper are as follows:

1. Using the example of rail irregularities, we summarize how noise is typically specified.
2. We then shortly define the probability distribution of the noise signals generated in this paper.
3. Starting from a given PSD we rigorously derive a way to shape this frequency content onto a noise signal. This method will turn out to be perfectly simple to use and to be applicable to almost any kind of noise spectrum.
4. We finally implement the approach and verify that it yields the same results as conventional methods.

2 Railway track irregularities

Besides safety and operating efficiency, it is an essential goal of railway vehicle design to provide an accepted level of vibration comfort. In order to take human perception into account, different methods and standards exist for passenger comfort assessment. However all of these rely on the accelerations experienced by the passengers as input information.

From a vehicle dynamical point of view these accelerations are the result of forced vibrations of the vehicle/track-system that are excited by track irregularities. Frederich (1984) analyzed a large number of track measurements and introduced representative PSDs for good, average and bad tracks, see Fig. 1. Note, these numbers quantify the irregularity per meter track length or with respect to the spatial frequency (unit: 1/m), respectively, and have to be transferred into the time domain taking the vehicle speed into account, see e.g. (Popp and Schiehlen, 2010).

Regarding the vehicle/track system that is excited by the track unevenness we confine ourselves to vertical dynamics and use the simplified quarter car model shown in Fig. 2. The excitation input is introduced as a variable track height z defined as a stochastic function of the longitudinal track position. The wheel/rail contact is represented by a stiff but linear spring/damper system. Rail and its support constitute a dynamical subsystem on the track side of the model, suspension and car body form the vehicle subsystem. The acceleration of the car body a is the output quantity of the model. The resulting Bode diagram is depicted in Fig. 3.

Here, the model is defined linear by intention. Presuming a constant running speed of the vehicle, the acceleration response of the car body can be evaluated in the frequency domain (Knothe and Stichel, 2003, Ch. 6), which provides the opportunity of comparison and validation with results from time domain simulations in Modelica. Fig. 4 presents the pure frequency domain results, that are based on the excitation by a track of all three qualities.

The results presented in this paper are confined to linear systems in order to validate the time-domain simulation approach with a well know frequency domain solution. However, this limitation can be dropped, once the results from time domain simulations with appropriately shaped noise spectra is validated. Time domain simulations are then available for non-linear systems, are capable of running with variable speed and may consider singular disturbances such as running over railway switches as well.

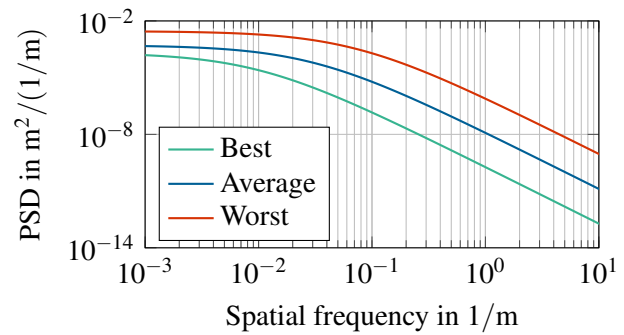


Figure 1. Representative track irregularity PSDs (Frederich, 1984).

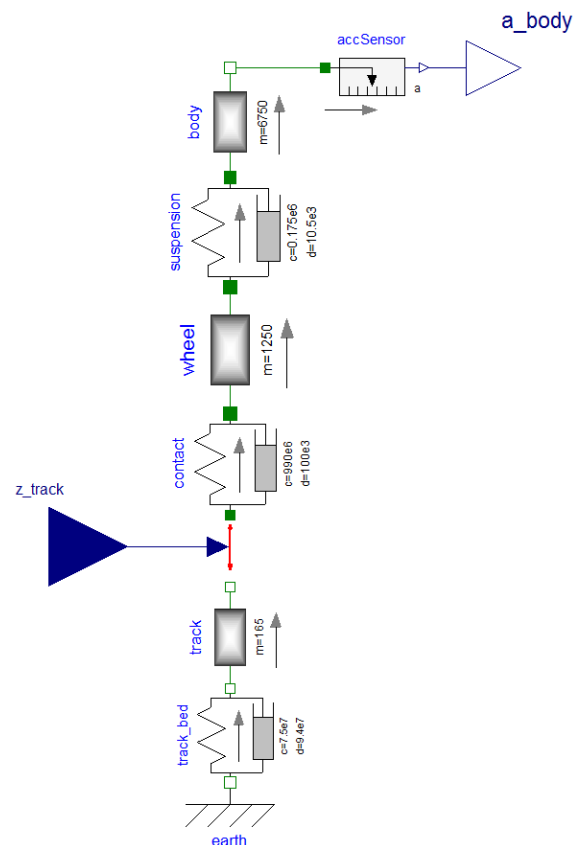


Figure 2. Simplified quarter car model of a railway vehicle in Modelica.

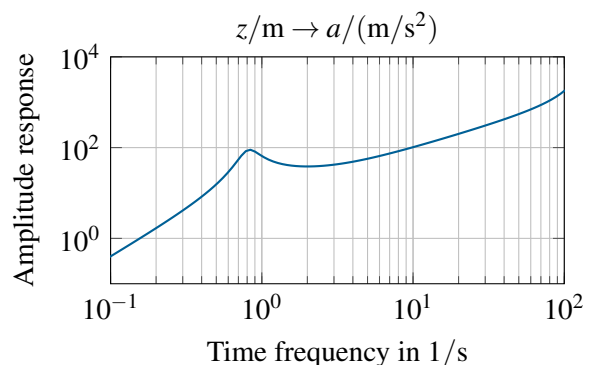


Figure 3. Amplitude response from track irregularity (in m) to body acceleration (in m/s²).

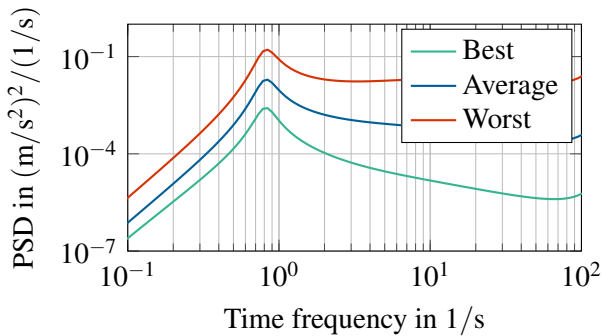


Figure 4. PSD of the body acceleration a for different track irregularities at a velocity of $v = 100$ m/s

3 Review of Noise parameters

The general degrees of freedom in parameterizing noise have been described in detail earlier (Klöckner et al., 2014). A noise signal can be specified in three steps:

1. Select a random number generator, which generates uniformly distributed random numbers with certain statistical properties, such as subsequent numbers being independent from each other.
2. Transform the uniformly distributed random numbers in order to match a given probability density function, such as for a normal distribution.
3. Interpolate the resulting stream of correctly distributed random numbers.

The random number generators of the `xorshift` family (Vigna, 2014) have been included in the `Noise` library since its last release.¹ These generators have very strong statistical and computational properties and are thus used in the library without exception.

In all cases, where unsampled random numbers are required, the `DIRCS` generator is used. This random number generator does not require a state but generates a random number directly from a double input signal. To this end, the `xorshift64*` algorithm is first initialized with the double input signal casted to two integer values. After ten iterations, the output of `xorshift64*` is used to seed an `xorshift128+` generator for a final iteration. In this way, high quality random numbers are produced by the efficient `xorshift` generators with a low computational effort for arbitrary input values.

The standard normal distribution is chosen for all random numbers generated in this work. This does not allow to reproduce effects commonly found in measurement noise, such as discretization. However, the choice is reasonable when complex filters are used to shape the actual noise signal to be used in the simulation. Typical filter parameterizations for rail irregularities

e.g. assume standard normal distributions of their input signals (SIM, 2003). Additionally, the subsequent interpolation relies on computing the weighted sum of consequent random numbers. Following the central limit theorem, the result will inevitably be shaped towards the normal distribution.

In previous work, we have described three distinct interpolation functions for noise signals. These include piece-wise constant and linear interpolations as well a smooth interpolation using the sinc function. The interpolations yield a continuous-time random signal $r(t)$ by computing the sum of consequent random numbers w_i , weighted with an admissible kernel function $k(t)$:

$$r(t) = \sum_{-n}^{+n} w_i \cdot k(t - i\Delta t), \quad (1)$$

with

$$k(i\Delta t) \stackrel{!}{=} \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{if } i \neq 0. \end{cases} \quad (2)$$

In this equation, Δt is the sample period of the random numbers and the interpolation base n has to be chosen according to the selected kernel $k(t)$.

However, for the more general case of a given PSD, the final interpolation step has to be replaced by a more powerful approach as described in the following sections.

4 Application of a given PSD

As already explained in Section 2, instead of (band-limited) white noise, colored noise is required for most practical applications. The required frequency content of the noise signal is usually specified by a given PSD $\Phi(f)$. In order to apply this PSD to a raw white noise signal with piece-wise constant interpolation a linear form filter is typically applied (SIM, 2003, VIII-TE:8). This filter $H(f)$ defines a mapping in the frequency domain between the white noise input vector $w(t)$ and the colored noise output vector $r(t)$:

$$R(f) = H(f)W(f) \quad (3)$$

where $R(f)$ and $W(f)$ are the Fourier transforms (FTs) of $r(t)$ and $w(t)$. White noise is defined by its flat PSD of $W(f) \equiv 1$. If the filter $H(f)$ is applied to white noise, the PSD of the colored noise is thus simply

$$\Phi_r(f) = |R(f)|^2 \equiv |H(f)|^2. \quad (4)$$

In order to shape colored noise to a given PSD, the required filter is hence constrained by

$$|H(f)|^2 \stackrel{!}{=} \Phi(f). \quad (5)$$

In practice, the filter is typically applied by fitting a rational transfer function on $\Phi(f)$ which is then simulated as an additional linear block in the model (see Section 4.1). An alternative exploiting the interpolation kernel from Eq. (1) is proposed in Section 4.2.

¹<https://github.com/DLR-SR/Noise/tree/MSL>

4.1 Using a transfer function

Before the approximation of a given PSD with a rational transfer function is explained, important properties are briefly repeated. In principle, the filter $H(f)$ is restricted to rational functions which are usually expressed w.r.t. the Laplace variable $s = d + 2\pi j f$, i.e.

$$H(s) = \frac{N(s)}{D(s)} = \frac{\sum_{k=0}^{n_z} a_k s^k}{\sum_{l=0}^{n_p} b_l s^l}. \quad (6)$$

The coefficients a_k of the numerator $N(s)$ and the coefficients b_l of the denominator $D(s)$ are real numbers. Both, the numerator and denominator can be factorized which leads to

$$H(s) = \frac{\prod_{k=1}^{n_z} (s - z_k)}{\prod_{l=1}^{n_p} (s - p_l)}. \quad (7)$$

Every zero z_k and every pole p_l is either a real number or two zeros (or poles) are each a complex conjugate pair. A necessary condition to express $H(s)$ in a state space representation is that $H(s)$ is proper, i.e. the number of poles n_p is greater than or equal to the number of zeros n_z . If the real part of all poles and zeros is negative, a transfer function is called minimum phase.²

In addition to constraint (5), $H(s)$ must be proper and minimum phase, in order to be realizable by a state space system. Because a suitable transfer function $H(s)$ cannot be analytically computed from a given PSD $\Phi(f)$ in general, a least squares fit is performed:

$$\min_{a_k, b_l} \sum_i^{n_f} \left(|H(2\pi j f_i)| - \sqrt{\Phi(f_i)} \right)^2. \quad (8)$$

The coefficients a_k and b_l are chosen as decision variables because they are real numbers and independent from each other. In order to ensure that the filter is proper, $n_z = n_p - 1$ is chosen. The optimization is pursued with MOPS (see Joos et al., 2002) and a Levenberg-Marquardt algorithm is used.

Finally, the minimum phase requirement is fulfilled by a subordinate step. To that end, the zeros z_k and poles p_l of the optimal solution are computed. Afterwards, the real part of every pole/zero is mirrored into the left half plane, e.g.

$$\bar{p}_i = -|\Re(p_i)| + \Im(p_i). \quad (9)$$

Note that the latter operation alters only the phase but not the amplitude of $H(s)$.

²Minimum phase means that both the filter and its inverse are stable and causal.

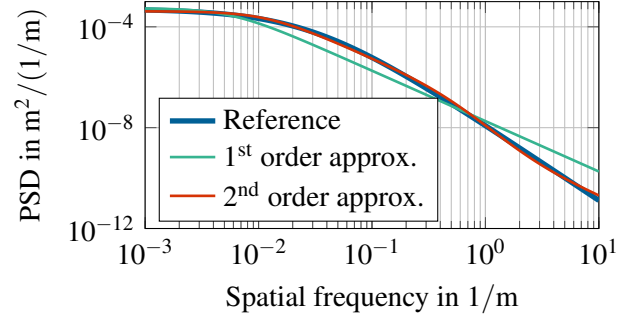


Figure 5. Approximation of the average track irregularity PSD with a first order and a second order filter. The second order filter shows a good fit to the reference.

Figure 5 compares the reference PSD to the PSDs of a first and second order filter. It can be seen that the second order filter is a very good approximation of the average track irregularity. This filter can thus be used to implement the form filter.

4.2 Using the interpolation kernel

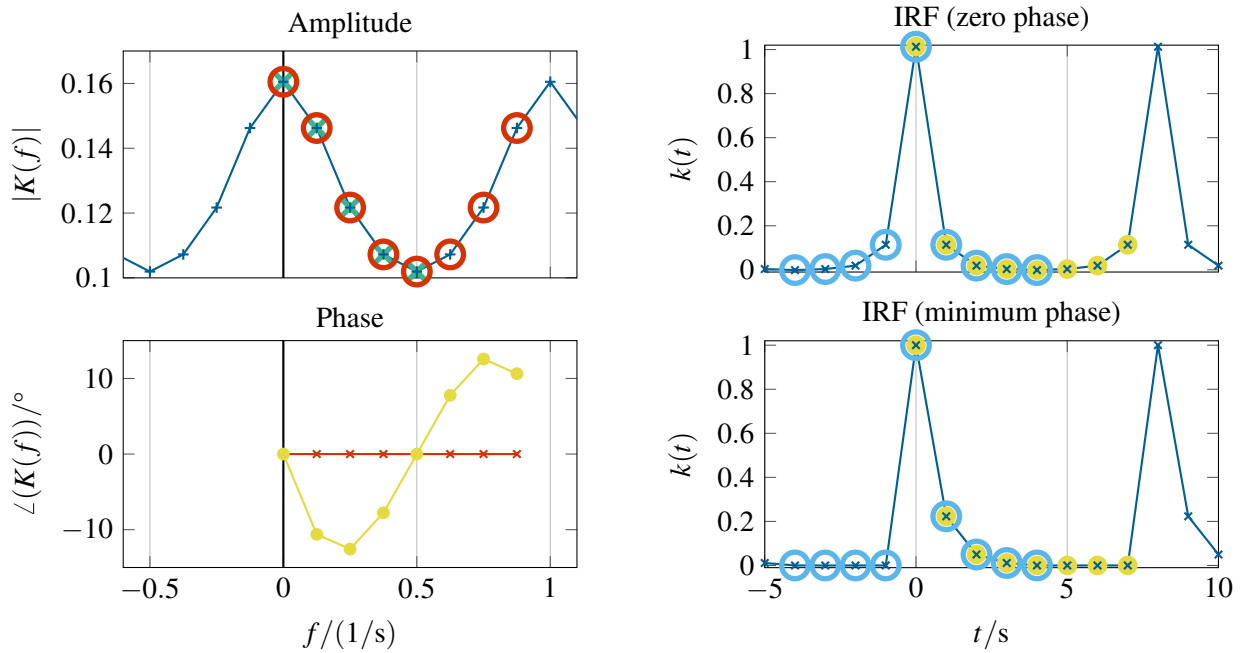
The filter or transfer function is typically implemented using continuous-time states in Modelica. This approach has two major drawbacks: First, expressing the filter in the time domain limits the simulation to a fixed velocity in order to map the location to a time-domain filter. Second, the additional states of the filter require the raw noise signal to be generated accurately using events, which considerably slows down simulation, even if only low accuracy is required. In this paper, we introduce a different approach to shaping the frequency content of the noise signal using the interpolation kernel $k(t)$ from Eq. (1).

4.2.1 Theoretical background

The idea is based on the convolution theorem. It relates the continuous-time integration of the filter states to a convolution integral of the raw noise signal $w_i(t)$ with the filter's IRF $h(t)$. Exploiting the piece-wise constant noise signal, this approach can be further reduced to a sum of weighted random numbers w_i :

$$\begin{aligned} R(f) &= W(f)H(f) \\ &\uparrow \\ r(t) &= w_i(t) * h(t) \\ &= \int_{-\infty}^{+\infty} w_i(t) \cdot h(t - \tau) d\tau \\ &= \sum_{-\infty}^{+\infty} \left(w_i \cdot \int_{i\Delta t}^{(i+1)\Delta t} h(t - \tau) d\tau \right). \end{aligned} \quad (10)$$

The weights in (10) are specified by the integral of the IRF $h(t)$. This integral can also be expressed as the step response $\zeta(t)$ of the filter. Assuming a stable filter,



(a) Frequency domain: In the upper plot, the selection of the correct samples for the iFT and Hilbert transform is illustrated. The given amplitude data (X) is mirrored at $f = 0$ and periodically repeated (—+—). Afterwards, the highlighted samples (O) are chosen. In the lower plot, the zero phase (—x—) and the minimum phase (—●—) are depicted.

(b) Time domain: The resulting IRF for the zero phase (upper plot) and minimum phase (lower plot) are depicted. The results from the iFT (●) are periodically repeated (—+—) in order to chose the correct samples (O).

Figure 6. The selection of the correct samples is illustrated in the time domain and in the frequency domain.

the convolution can finally be approximated using a truncated sum:

$$\begin{aligned}
 r(t) &= \sum_{i=-\infty}^{+\infty} w_i \cdot (\zeta(t-(i+1)\Delta t) - \zeta(t-i\Delta t)) \\
 &\approx \sum_{i=\lfloor t/\Delta t \rfloor - n}^{\lfloor t/\Delta t \rfloor + n} w_i \cdot \underbrace{(\zeta(t-(i+1)\Delta t) - \zeta(t-i\Delta t))}_{:=k(t)}. \quad (11)
 \end{aligned}$$

Using this approach, all continuous and discrete states can be eliminated from the noise generation. This was shown before to be advantageous for the simulation performance (van der Linden et al., 2015). Additionally, the interpolation kernel $k(t)$ can be shaped using an arbitrary filter, if its step response is known.

4.2.2 Computation of the IRF

As we have seen, only a suitable step response is required to shape the desired frequency content. This IRF can easily be obtained from the transfer function derived in Section 4.1. However, in order to avoid the approximation with a rational function, the IRF is directly computed from the PSD using the framework of Fourier transform (FT) and inverse Fourier transform (iFT). The resulting IRF is then numerically integrated in order to yield the step response.

Because the PSD describes only the amplitude of the filter and because the filter must be not realized in state

space representation it is possible to use the phase as an additional degree of freedom. Here, two different phases are considered: zero phase and minimum phase.

Zero phase: First the zero phase case is considered. For this case all phase are set to zero. The FT of the interpolation kernel is hence simply

$$|K(f)| = \sqrt{\Phi(f)}. \quad (12)$$

However, for a correct application of available FT algorithms, the frequency samples must be chosen carefully: In order to yield a real valued $k(t)$, $K(f) = \text{conj}(S(-f))$ must hold. It is further helpful to remember that – because time and frequency are both discretized – $K(f)$ and $k(t)$ are periodically repeated. This is illustrated in Fig. 6a for a simple example and the correct samples are marked.

After the iFT, the resulting $k(t)$ is periodically repeated, too. This allows to chose the correct samples as depicted in Fig. 6b. As it can be further seen, the zero phase yields a non-causal IRF which is symmetric to $t = 0$.

Minimum phase: Second, the minimum phase case is considered. In this case, only the amplitude of the FT of the interpolation kernel is given by the PSD:

$$|K(f)| = \sqrt{\Phi(f)}. \quad (13)$$

The minimum phase $\angle(K(f))$ can be computed using the Hilbert transform. For the Hilbert transform, the same samples must be chosen as for the iFT. The resulting minimum phase is depicted in Fig. 6a. Afterwards, the full FT of the interpolation kernel is given by

$$K(f) = |K(f)| \cdot \exp(j \cdot \angle(K(f))). \quad (14)$$

The transformation into the time domain is subsequently performed in the same way as for the zero phase case. As it can be seen in Fig. 6b, the minimum phase filter is causal, i.e. its IRF is non-zero only for non-negative times $t \geq 0$.

Figure 7 compares IRFs for the average track irregularities as obtained from the procedure outlined above. First, the IRF of the fitted second order filter is evaluated by simulation and by iFT of the PSD shown in Fig. 5. Both results are essentially the same, showing the correct iFT application. The IRFs obtained directly from the given PSD are also shown. The minimum phase IRF is very similar to the fitted filter's IRF, underlining the good fit of the filter. The zero phase IRF is non-causal, as it is non-zero for negative times.

5 Results

The form filters for average track irregularities are implemented using the `Noise` library according to the procedures outlined above. Using the Dymola 2016 RC2 simulation tool with its DASSL solver, the different steps of the implementation are then verified. To this end, the following simulation experiments are presented:

1. The minimum phase convolution is verified against a state space filter implementation with identical white noise input.
2. The white noise input of the state space implementation is exchanged by an independent noise source not using the DIRCS algorithm.
3. The minimum phase and zero phase IRFs are compared to each other.
4. The quarter car model of the railway vehicle is fed with the zero phase noise convolution filter and compared against the standard frequency domain solution.

5.1 Convolution verification

Figure 8 shows a comparison of the fitted second order filter's state space implementation with its minimum phase convolution implementation. Both filters are driven by identical white noise generated with the DIRCS generator directly from the position on the track. The raw random numbers are generated with a sample period of $\Delta x = 0.4\text{m}$ and a standard deviation of

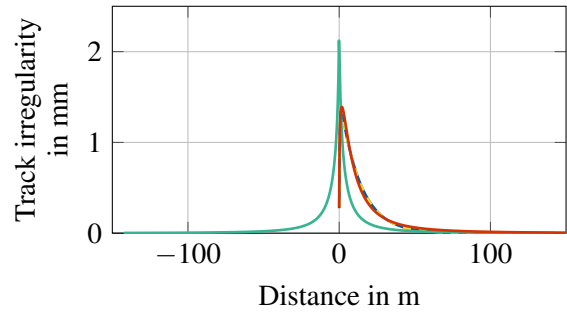


Figure 7. Comparison of different IRF: The IRF of the 2nd order filter is obtained by simulation (—) and by iFT (---). An excellent agreement can be recognized. Additionally, the minimum phase (—) and zero phase (—) IRF are depicted. These correspond directly to the specified PSD.

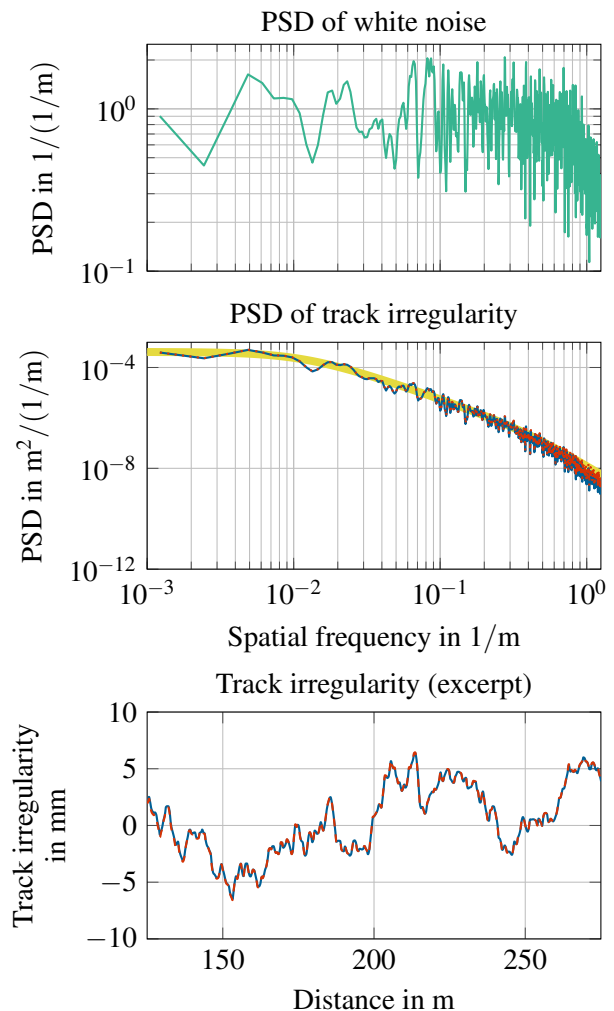


Figure 8. The identical white noise (upper plot) is applied to the second order filter by simulation (—) and by convolution (⋯). The PSD of both implementations (middle plot) are identical and correspond well with the reference PSD (—). As it can be seen in the lower plot, the signals are also identical.

$\sigma = \sqrt{0.5/\Delta t}$. It can be seen that the white noise spectrum has indeed a spectral density of 1; except for its random nature. The shape of the white noise spectrum directly corresponds to the shape of the track irregularity PSDs. Both irregularity PSDs can be seen to be identical. This is also the case for the actual simulation output z of the track irregularity.

5.2 Space domain noise verification

In the second step, the minimum phase convolution implementation is compared to a traditional state space implementation fed by a time domain sampled noise. In order to yield comparable results, the time domain noise is sampled with $\Delta t = \Delta x/v$. The standard deviation of the normal distribution is kept the same. Figure 9 compares both results to the reference. Good agreements can be observed in both time frequency domain and spatial frequency domain PSDs. Timing comparisons are not presented, as the current convolution implementation is not yet optimized.

5.3 Minimum and zero phase filters

The comparison of the minimum phase and zero phase convolution implementations can be seen in Fig. 10. The white noise generator is DIRCS in both cases and the sample periods are both $\Delta x = 0.4\text{m}$. Both IRFs have a resolution of 0.1m . Both variants agree very well with the reference in the low frequencies. At very high frequencies, the influence of the two sampling periods are marked with vertical lines. Characteristic marks on the PSDs can be seen at these lines. The respective sampling periods must thus always be chosen high enough to resolve all relevant effects.

5.4 Quarter car railway vehicle

Finally, the zero phase convolution implementation is integrated with the quarter car model of a railway vehicle running at $v = 100\text{m/s}$. Figure 11 compares the acceleration of the car body from this simulation to the well trusted frequency domain solution. A very good agreement can be seen.

6 Conclusions

In this paper, we show how to consistently shape a noise signal to match a given frequency content specified by a PSD. The method is introduced at hand of the quarter car model of a railway vehicle, for which well trusted reference solutions are available.

Our method uses the non-recursive random number generator DIRCS. It generates normally distributed random numbers directly from the current location on the track. We then shape the frequency content of the random numbers using a convolution with the impulse response of a form filter. It is shown that the IRF of the form filter can be directly generated from the given PSD using the inverse Fourier transform.

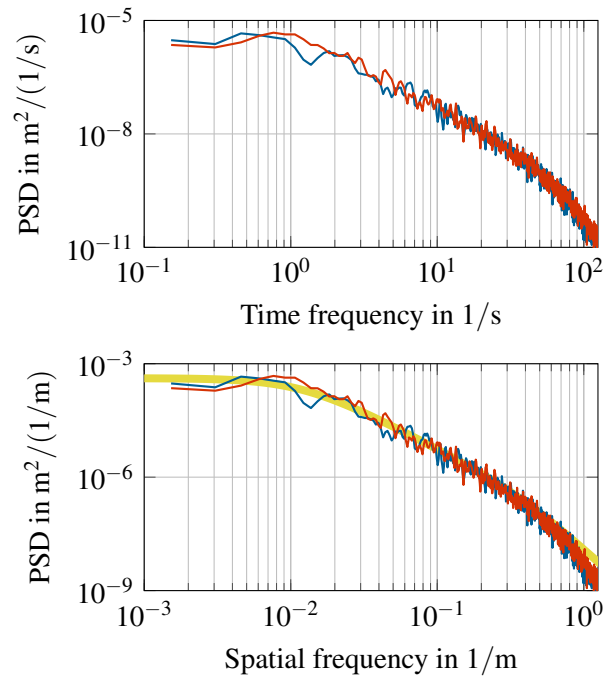


Figure 9. The PSD of the track irregularity from simulation (—) and from convolution (—) are compared in the time frequency domain (upper plot) and in the spatial frequency domain (lower plot). Both agree very well. In the spatial frequency domain, an excellent agreement with the reference (—) can be also seen.

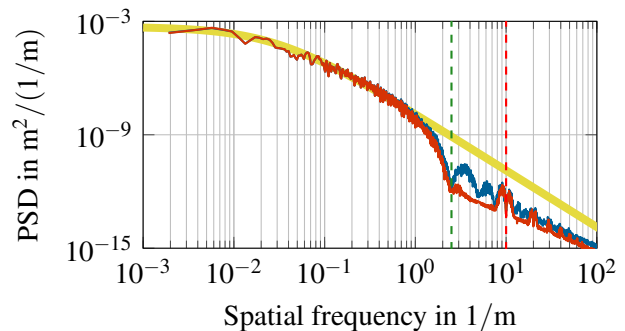


Figure 10. The PSD resulting from the minimum phase (—) and from the zero phase (—) iFT agree well with the reference (—).

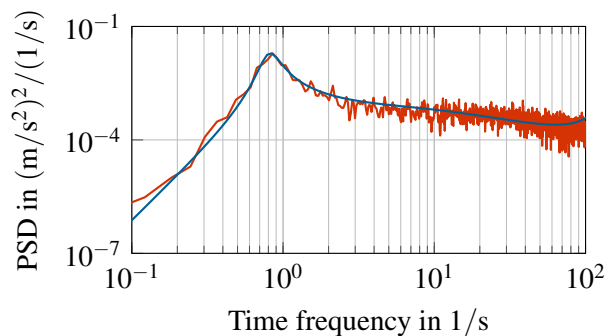


Figure 11. The frequency domain solution is compared (—) to simulation results (—).

The method is implemented in the `Modelica Noise` library and validated against the given spectrum. It is then used to excite a linear quarter car model running at a constant speed. Results obtained with our method agree very well with the standard solutions based on frequency domain computations.

Our method is thus shown to produce correct results. Additionally, it can be employed not only for linear models, but also for non-linear models, vehicles running at varying speed, or in more complex scenarios involving e.g. singular disturbances. Moreover, the method proposed in this paper is perfectly straightforward to use and can also be applied to a variety of problems, such as turbulence or street roughness.

The current implementation is based on the open-source `Noise` library. This makes the method available to a wide audience and also gives room for further improvements. Since our implementation of the convolution has not yet been optimized, further investigations should take into account timing measures. Extensions of the method could possibly be found in higher dimensional noise spectra, correlated noise, or additional effects such as discretization.

Reproducible research

The results of this paper can be reproduced using the code which will be made available on <http://dlr-sr.github.io>.

Acknowledgments

We thank our parents for giving us a great first name.

References

- EASA CS-25. Certification specifications and acceptable means of compliance for large aeroplanes, 2013.
- Fritz Frederich. Die Gleislage aus fahrzeugtechnischer Sicht. *ZEV-Glasers Annalen*, pages 108–1984, 1984.
- Joachim Haase, S Wolf, and C Clauß. Monte carlo simulation with modelica. In *6th International Modelica Conference*, pages 601–604, Bielefeld, Germany, 2008.
- H.D. Joos, J. Bals, G. Looye, K. Schnepfer, and A. Varga. A multi-objective optimisation-based software environment for control systems design. In *Conference on Computer-Aided Control Systems Design*. IEEE, 2002.
- Andreas Klöckner, Franciscus L. J. van der Linden, and Dirk Zimmer. Noise generation for continuous system simulation. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings of the 10th International Modelica Conference*, number 96 in Linköping Electronic Conference Proceedings, pages 837–846, Lund, Sweden, March 10-12 2014. Modelica Association and Linköping University Electronic Press. doi:10.3384/ECP14096837. ISBN: 978-91-7519-380-9. ISSN: 1650-3686. eISSN: 1650-3740.
- K. Knothe and S. Stichel. *Schienenfahrzeugdynamik*. Springer, Berlin, 2003.
- Hans Wolfgang Liepmann. On the application of statistical concepts to the buffeting problem. *Journal of the Aeronautical Sciences (Institute of the Aeronautical Sciences)*, 19(12), 1952. doi:10.2514/8.2491.
- Alexander Pollok, Dirk Zimmer, and Francesco Casella. Fractional-order modelling in modelica. accepted and to be presented at the 11th International Modelica Conference, 2015.
- K. Popp and W. Schiehlen. *Ground Vehicle Dynamics*. Springer, 2010.
- SIMPACT Time Excitations Catalogue*. SIMPACK, 24th September 2003. SIMDOC v8.607.
- Franciscus L. J. van der Linden, Andreas Klöckner, and Dirk Zimmer. Effects of event-free noise signals on continuous-time simulation performance. In Felix Breiteneker, Andreas Kugi, and Inge Troch, editors, *8th Vienna International Conference on Mathematical Modelling*, volume 8 of *Mathematical Modelling*, pages 280–285, Vienna, Austria, 2015. IFAC-PapersOnLine. doi:10.3182/20150218-3-AU-30250.
- Sebastiano Vigna. Further scramblings of marsaglia’s xorshift generators. *CoRR*, abs/1404.0390, 2014. URL <http://arxiv.org/abs/1404.0390>. Code available at <http://xorshift.di.unimi.it/>.

Dynamic Modelling of a Flat-Plate Solar Collector for Control Purposes

Saioa Herrero López Susana López Pérez Itzal del Hoyo Arce Iván Mesonero Dávila

IK4-TEKNIKER

Parke Teknologikoa, Iñaki Goenaga 5, 20600 Eibar (Spain)

{saioa.herrero, susana.lopez, itzal.delhoyo, ivan.mesonero}@tekniker.es

Abstract

Two different dynamic models of a flat-plate solar collector have been developed in the Modelica language under Dymola[®] software.

These models have been developed within the Ambassador Project (Onillon, 2014). In this project, models of district heating components are conducted for control purposes, including a solar plant model.

The present article describes in detail each of these models along with the development process (e.g., assumptions taken into account). The model validation process and results are also presented, as well as the corresponding discussion and conclusions. The model's validation has been conducted by comparing the model's simulation results with the experimental results obtained in the IK4-TEKNIKER Solar Thermal Test Rig.

Keywords: Solar collector, Dynamic model, Control design, Modelica

1 Introduction

Solar water heating has received increased interest in recent years, primarily because it is a free energy source, and it is available, in principle, anywhere all over the world.

The key element in a solar heating plant is the solar collector field, as it is at the solar collectors that the solar energy is captured and transferred to the circulating fluid. Currently, the collector type most widely used in such plants is the flat-plate solar collector.

Heating demand coverage involves not only a certain quantity of heat energy but also a specific water temperature. Besides, in the case of solar plants, the energy source is non-manageable; therefore, it is even more difficult than in conventional plants to assure required supply conditions.

Because of that, well-developed control is essential in this type of facility to allow the fulfilment of supply requirements, which will depend on the application.

One of these applications is a solar water heating plant connected to a district heating system. This scenario is covered by the Ambassador Project. The

core of this project is the development of suitable management by control algorithms, which assure optimum performance of the whole district energy system. Control design requires knowing in detail the physical behaviour of the system to be controlled; therefore, models of all the subsystems in the District Heating System, including the solar plant, are required.

Solar collectors are usually described by stationary models that consider the collector to be in steady-state operation (Hottel and Woertz, 1942; Hottel and Whillier, 1955; Bliss, 1959). Stationary models have the advantage of being simpler and hence needing less computation time than dynamic models. However, this simplification may be critical because solar collectors rarely reach a steady state during operation due to their large time constants and the variability of the driving forces. For several applications, e.g., the investigation of control strategies, it is desirable to take the collector dynamics into account (Schmieders, 1997; Ron, 1980).

Therefore, within this context, two flat-plate solar collector dynamic models have been developed in the Modelica language under the Dymola[®] environment: a Detailed Model and a Simplified Model.

2 The Detailed Model

Dynamic solar collector models can be classified into single capacitance (one node) models (Duffie and Beckman, 1991; Close, 1967), fluid flow direction distributed models (or 1xN node models) (Isakson, 1995; Muschaweck and Spirkl, 1993; Prapas, Norton et al 1988), and fluid flow direction and transverse distributed (or MxN node) models. Models of the last type try to represent the physical system in a more realistic way: apart from taking into account fluid temperature nodes (fluid flow direction), several temperature nodes at transverse directions are also used, with each one representing a solar collector component. There are several models developed in this way: 2x1 node (Klein, Duffie et al 1974), 2x node (Huang and Wang, 1994), 3x node (Ron, 1980), 4x node (Kammaing, 1985), or even more complicated ones (Oliva *et al*, 1991; Cadafalch, 2009).

In the present case, the so-called "Detailed Model", a 5x1 model, has been developed. On the one hand,

transverse nodes represent the glass cover, the air inside the collector, the absorber, the fluid, and the rear insulation. Regarding the longitudinal discretization (fluid flow direction), at first a $5 \times n$ distributed model ($8 > n > 1$) was chosen, but it was determined that the discretization level increase in the fluid flow direction did not yield a relevant difference; therefore, it was discarded in favour of model simplicity.

In the model, the following heat flows are considered between the collector components:

- Convection between the glass cover and the air in the gap. For this case, a natural convection is considered as well as between this air and the absorber plate. The equations correspond to a free convection between a fluid and a plate, taking into account the possibility of the fluid being hotter than the solid and vice versa, and the tilt angle of the collector (Chapman, 1987)
- The fluid flow through the absorber plate is modelled via a forced heat convection imposed between the fluid lines and the absorber, taking into account both laminar and turbulent regimes (Verein Deutscher Ingenieure, 1997).
- Radiation heat transfer has been taken into account between the glass cover and the absorber plate.
- Conduction heat transfer between the absorber and the rear insulation is modelled (conduction in a Plane Wall) (Chapman, 1987).

In addition, heat transfer phenomena are also considered between the collector and the ambient, and has been modelled under the following conditions:

- Radiation between the glass cover and the sky. The model calculates the sky temperature from the value of the clearness index, the relative humidity and the ambient temperature.
- Both forced (wind action (Sartori, 2006)) and natural convections (same equations as for the collector internal convection) between the glass cover and the ambient are included. In the case of the rear insulation, only natural convection has been modelled.
- The solar radiation reaching the collector is applied to both the absorber and glass cover energy balances. For the glass cover, the element absorptivity is included as a parameter. With respect to that reaching the absorber, the glass cover transmissivity and the absorber absorptivity are taken into account, with both depending on the incidence angle of the solar radiation.

The scheme in Figure 1 represents the conceptual idea of modelled heat transfer phenomena.

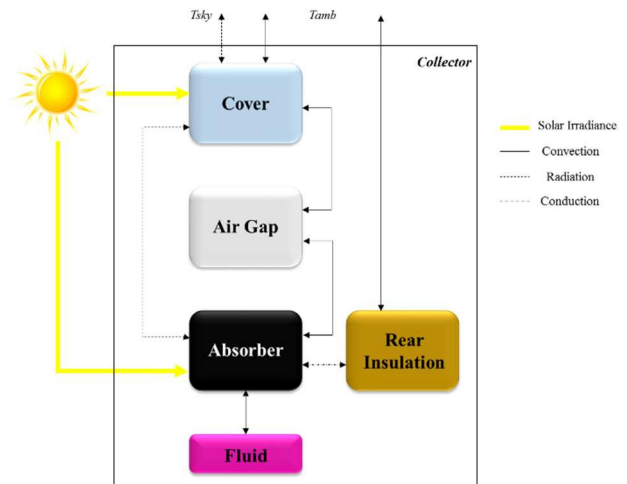


Figure 1. Schematic of heat transmission phenomena modelled in the Detailed Model

Following, physical assumptions associated with this type of model and those considered when developing the model are collected:

- All heat transport phenomena are taken to be in 1-D perpendicular to the flow direction, except for the heat carried by the flow. Perfect insulation is considered at the edges, so all heat transfer phenomena are related to the frontal collector area.
- With respect to the absorber, a harp type has been chosen; i.e., in each collector the fluid flow is distributed through a certain number of parallel tubes. The fluid flow is considered uniform along all the tubes in the absorber.
- Modelled pressure losses are those taking place at the collector harp tubes. Neither pressure losses at the input/output of the collector nor at the manifolds are modelled. This is because it was considered not important to develop specific detailed models for those elements. If real pressure losses need to be included, the best way to do it is by including data obtained from experimental tests (see Simplified Model in Section 3).
- Natural convection is considered between the rear insulation component and the ambient.
- As mentioned before, the solar energy not only affects the energy balance of the absorber plate (taking into account the glass cover's transmissivity and the absorber's absorptivity mentioned below) but also that of the glass cover itself.
- The cover's solar absorptivity is considered constant (independent of temperature and solar spectrum). However, the cover's transmissivity and IR absorptivity are dependent on incidence angle.
- Only fluid properties depend on temperature. The rest of the components' physical properties are not dependent on this variable. If desired, the components' properties, such as thermal inertias,

can be easily turned into temperature-dependent properties by replacing the elements with those in the NewThermal Library (López, Hoyo, 2014).

A picture of the corresponding Modelica model built under the Dymola® environment can be observed in Figure 2.

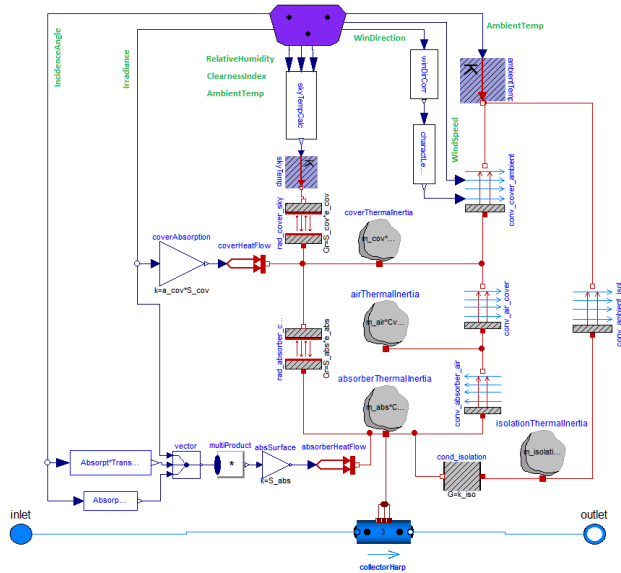


Figure 2. Solar Collector Detailed Model developed under Dymola® software

As shown in Figure 2, the Detailed Model is built with models from the Modelica Standard Library, mainly, HeatCapacitor, ThermalConductor, Convection, BodyRadiation, and DynamicPipe.

Each of the primary components in the solar collector (glass cover, air inside the gap, absorber plate, and rear insulation) is modelled as a thermal inertia with a certain uniform temperature finite volume (HeatCapacitor).

The fluid model itself represents an incompressible fluid and it is characterized by the value of its main properties, such as density, viscosity, specific heat capacity, conductivity and vapour pressure. Any fluid model included in the Modelica Standard Library can be used (DynamicPipe selector), but for the validation phase (see Section 4), a specific fluid model (Tyfocor LS) has been developed.

To simulate the behaviour of the collector when exposed to the solar radiation with a fluid circulating inside via the Detailed Model, apart from the model parameters, the following inputs are needed:

- Irradiance in the plane of the collector
- Incidence angle of solar radiation
- Clearness index
- Relative humidity
- Ambient temperature
- Wind velocity and direction

- Fluid input port conditions (i.e., mass flow rate and temperature)

It can be observed that the model's discretization level is high, so it can be easily adapted to other solar collector designs. To achieve this, it includes many parameters (e.g., geometries and various material physical properties), inputs and state variables.

3 The Simplified Model

As mentioned, the described Detailed Model has a significant number of parameters, and many of them are related to geometrical and thermal properties of the materials, which are not usually provided by the manufacturer. Therefore, it could be difficult to set up the Detailed Model.

The purpose of the Simplified Model is to develop a workable model; i.e., one that provides the most representative solar collector dynamics using the minimum number of parameters that are easy to obtain, with a minimum simulation time.

Regarding the model's parameters, several standards have been developed to normalize the solar collector's performance data via solar thermal collector testing. Historically the US ASHRAE standard (93-77) was the first one to be widely used. Next, the ISO 9806 series of standards was developed and from them, the EN 12975. Several national standards are also available outside of Europe, most often based on the ISO 9806, but in Europe the EN 12975 has replaced all national standards.

Taking as a reference EN12975, it distinguishes between steady state test conditions and quasi-dynamic test conditions. Currently the most common tests between manufacturers are those performed under steady state conditions.

$$\dot{Q}/A = F'(\tau\alpha)_{en}G^* - c_1(t_m - t_a) - c_2(t_m - t_a)^2 \quad (1)$$

Equation (1) represents the static behaviour of a flat-plate solar collector according to EN12975. The standard describes tests for working out all of the parameters in the equation: those related to the heat reaching the fluid (F'), and those related to collector thermal losses (c_1 , c_2). Currently there are a couple of Modelica libraries including solar collector models based on equation (1)¹.

However, within the standard stationary tests' descriptions, additional optional test procedures are included, which is the case for the effective thermal capacity (c_5) and the incidence angle modifier ($K_{\theta b}(\theta)$). Including these two additional parameters results in the

¹ AixLib library (SolarThermal model), and Building Systems library (ThermalCollectorDynamic model)

following equation representing the collector, also included in the referenced standard:

$$\begin{aligned} \dot{Q}/A = & F'(\tau\alpha)_{en}K_{\theta b}(\theta)G^* - c_1(t_m - t_a) \\ & - c_2(t_m - t_a)^2 - c_5 \frac{dt_m}{dt} \end{aligned} \quad (2)$$

Related to the effective thermal capacity, it involves an equivalent collector global thermal capacity, lumping into one temperature node the heat capacities of all the collectors' components. By simply adding this parameter to equation (1), it becomes dynamic, resulting in a solar collector model type called a Single-capacitance (or one node) model.

In this way, the dynamic behaviour is considered, not in a detailed way but adequately for use of the model for control design purposes.

Therefore, equation (2) is chosen as the model basis for the Simplified Model. Currently there is a Modelica library already including a dynamic solar collector model that takes into account this thermal capacity; however its approach is different from that shown at equation (2)².

It must be noted that in equation (2), mean fluid temperature (t_m) is considered the arithmetic mean between the inlet and outlet temperatures of the collector. This finding implies theorizing a linear temperature distribution in the collector, as several authors did previously (Close, 1967). However, according to other authors (Duffie and Beckman, 1991), in reality, this distribution is not observed. In the EN 12975 standardized test the arithmetic mean is used; therefore, because this standard is the reference for the Simplified Model, the arithmetic mean will also be used in this case.

Equation (2) actually represents a thermal energy balance, where the following terms appear:

- The useful energy gain, as heat energy absorbed by the fluid (\dot{Q}), In the equation it appears as energy per unit collector area (A).
- Solar radiation absorbed by the collector, which depends on the collector's efficiency factor (F'), the effective transmittance-absorbance product for normal incidence ($(\tau\alpha)_{en}$), the incidence angle modifier for beam radiation ($K_{\theta b}(\theta)$), and the global hemispheric solar radiation (G).
- Energy losses, calculated according to parameters obtained from the standardized tests (c_1 , c_2), ambient temperature (t_a), and fluid arithmetic mean temperature (t_m).
- Energy storage, calculated from effective thermal capacity (c_5), and fluid arithmetic mean temperature variation.

Related to hydraulic behaviour, fluid pressure losses are also included in the model. The measurement of the collector pressure loss, although optional, is included in the referenced standard tests. Therefore, an ad-hoc model has been developed (`pressureLoss`) and included in the model to apply the corresponding total pressure losses according to the data obtained from the tests. This component initially applies the least squares method to approximate the test data to a 2nd order polynomial function without an independent term, for later calculation of the pressure drop depending on the flow rate during the simulation.

A picture of the corresponding Modelica model built under Dymola[®] environment can be observed in Figure 3.

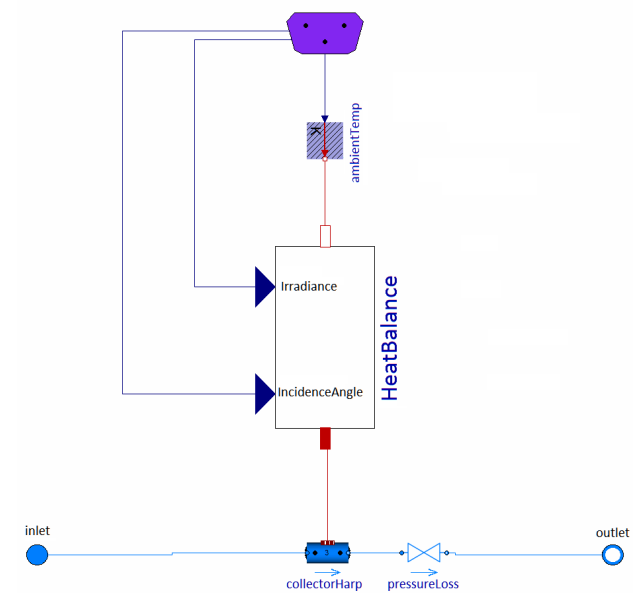


Figure 3. Solar Collector Simplified Model

In the previous figure, `HeatBalance` model is used to compute the heat flow calculation according to equation (2) and transmit it to the fluid (`collectorHarp DynamicPipe`). Despite 3 nodes being defined at the `collectorHarp`, the calculated heat flow is only applied to the central one. The extreme nodes are only used to calculate the mean fluid temperature.

In the case of the Simplified Model for the simulation, when exposed to the solar radiation with a fluid circulating inside, just the following inputs are needed:

- Irradiance in the plane of the collector
- Incidence angle of solar radiation
- Ambient temperature
- Fluid input port conditions (i.e., mass flow rate and temperature)

3.1 Series and Parallel configuration

In both centralized and decentralized solar plants, solar collectors are usually connected together in series,

² Buildings library (ASHRAE93 model and EN12975 model)

parallel or a combination of series and parallel arrangements.

To simulate the whole solar field would require the development of the corresponding simulation models based on the connections of the individual solar collector models, which would be very tiresome. Because of that, based on the described Simplified Model, two additional models have been worked out for the simulation of series and parallel arrangements of collector modules, respectively.

In a parallel-connected collector array, the flow of the heat transfer fluid is divided and a proportion goes through each collector in the array. This mainly involves the same pressure drop and the same temperature increment, and thus the same collector efficiency. Facing modelization coming from the Simplified Model, the parallel connection of N solar collectors is equivalent to a unique solar collector with an area N times larger, an N times higher heat capacity, an N times higher number of tubes, and the same pressure drop. In this new model (`ParallelArray` model), apart from the simplified collector model parameters, the user only needs to set up the number of solar collectors in the collector array.

Conversely, in a series-connected collector array, all of the heat transfer fluid passes through all of the collectors. This transfer primarily involves a pressure drop and output temperature increase; thus, collector efficiency decreases in the fluid flow direction. In this case the developed model is equivalent to a unique solar collector with the same number of tubes but N times longer, discretized into $N+2$ nodes in the flow line (`SeriesArray` model). The net heat flow calculated at each solar collector (balance of heat gain, heat losses, and heat storage) is applied to each one of the N central nodes. In the `SeriesArray` model, apart from the Simplified Model parameters, the user has to set up the number of solar collectors in the collector array, and the percent of additional pressure drop with respect to the theoretical array pressure drop (N times the individual solar collector pressure drop) due to the additional pipe length for the connections.

To check the developed models, simulations have been carried out in Dymola®, and simulation results have been compared between these new models and the corresponding assembly of individual simplified collector models. The checking has been carried out for the $N=3$ case, under certain ambient and fluid conditions (see Case B conditions at Section 4). The comparison of the obtained collector array outlet temperatures for the parallel and series connection is shown at Figure 4 and Figure 5, respectively.

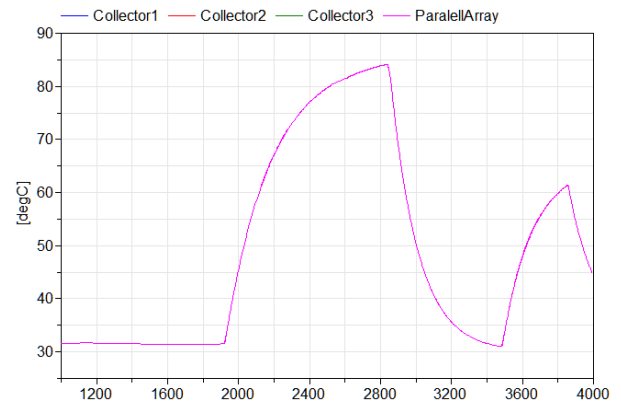


Figure 4. Outlet temperature comparison for the parallel arrangement configuration.

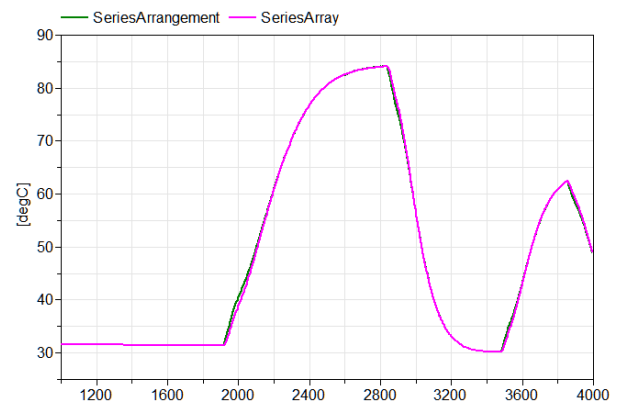


Figure 5. Outlet temperature comparison for the series arrangement configuration.

As expected, in the parallel configuration all outlet temperatures coincide perfectly: between individual collectors, and also with the `ParallelArray` model.

In the case of the series arrangement, the length of each flow control volume (node) of the `SeriesArray` model ($N+2 = 5$ nodes) is different from that of the series connection of 3 individual collectors ($3N = 9$ nodes). This involves small differences between the outlet temperatures, as indicated in Figure 5, which are more obvious in the transient states (error $<1\%$). It is considered that based on the small error obtained, the developed model behaviour is good, and it is not worth increasing the number of control volumes in the `SeriesArray` model.

4 Validation of the Detailed and Simplified Models

Two flat-plate solar collectors' dynamic models have been developed: the Detailed Model, and the Simplified Model. The validation of these models is carried out by comparing experimental data with the models' simulation results of a specific flat-plate solar collector working under certain operating conditions.

Experimental data are obtained from the Solar Thermal Test Rig located at IK4-TEKNIKER facilities (LER), normally used for the characterization of solar

thermal components, such as flat-plate solar collectors. This facility has a fully sensorized and actuated solar thermal installation, including a mini-weather station to collect meteorological data. In this facility, the fluid comes out of a DHW tank and is driven by the primary pump to the collector. Before reaching that point, the fluid passes through some heaters that increase the fluid temperature up to the desired value. The fluid makes its way to the collector where it is heated by solar radiation, and finally it goes back to the DHW tank, thus closing the circuit.

For validation, an appropriate environment has been developed in Dymola® for the Detailed Model and the Simplified Model simulations, to simulate their behaviour when working under ambient conditions with a fluid passing through. Among other things, this included the implementation of ad-hoc calculation models needed to turn available experimental data into the required input data for the Simplified and Detailed models.

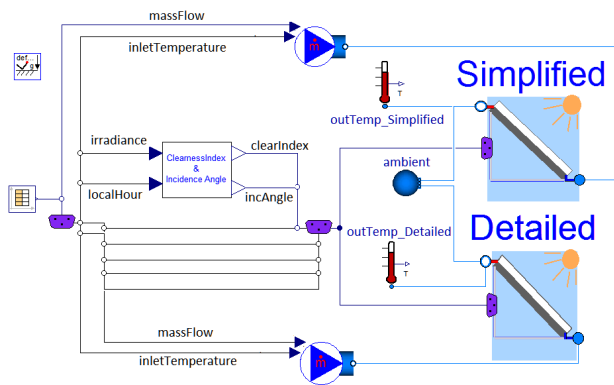


Figure 6. Dymola® environment for the Simplified Model and the Detailed Model simulations.

Regarding the models' parameters values, in the case of the Simplified Model they were obtained from the corresponding collector performance test report carried out according to UNE-EN 12975-2:2006 by a testing laboratory accredited by ENAC. In the case of the Detailed Model, most of the parameters' values were collected from the collector manufacturer's technical data sheets, and the rest (e.g., physical properties of certain materials) from specialized literature (Duffie and Beckman, 1991; Chapman, 1987).

In this type of plant, the heat transfer fluid degrades over time, so its properties values also change. In this case, when experimental tests for validation were carried out in the LER, the fluid was not fresh; therefore, in pursuit of representativeness/veracity in the simulation results, real physical properties were used for the fluid model definition instead of those coming from the technical data sheets. For that purpose, a fluid sample was removed from the circuit and characterized just after performance of the tests. Thus, reliable values for quantities such as density, thermal conductivity, specific heat capacity and

dynamic viscosity depending on temperature value were available for the simulations.

The aim of the validation process is to check the capability of the developed models to represent collector behaviour in general, and dynamic performance specifically. For that purpose, the following three collector operation cases have been established:

- Case A: mass flow rate variation (increasing steps), for almost constant incident solar radiation and inlet temperature values.

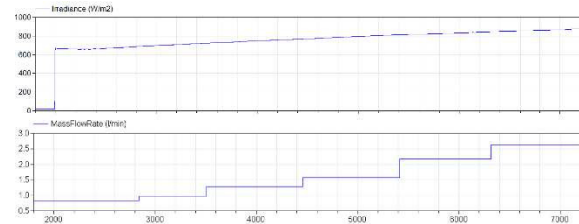


Figure 7. Case A mass flow rate profile.

- Case B: incident solar radiation with sudden variation (covered/uncovered), for different mass flow rates, with an almost constant inlet temperature

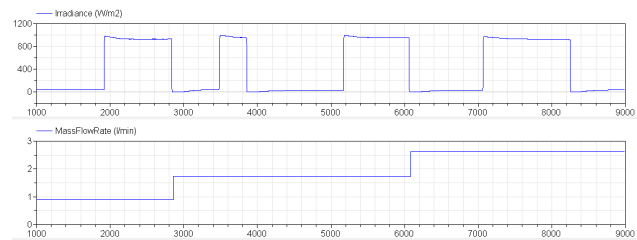


Figure 8. Case B irradiance and mass flow rate profiles.

- Case C: increasing inlet temperature, with decreasing incident solar radiation (not controlled) and almost constant mass flow rate

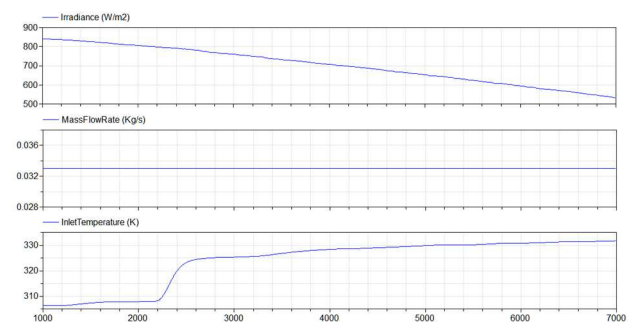


Figure 9. Case C Irradiance, mass flow rate, and inlet temperature profiles.

These cases allow the analysis of the models' responses to the variation of the possible controlled variables in this type of plant (mass flow rate and inlet temperature), and the main non-manageable perturbation (solar irradiance).

The resulting collector's outlet temperature value over the time at each case is shown in the following figures.

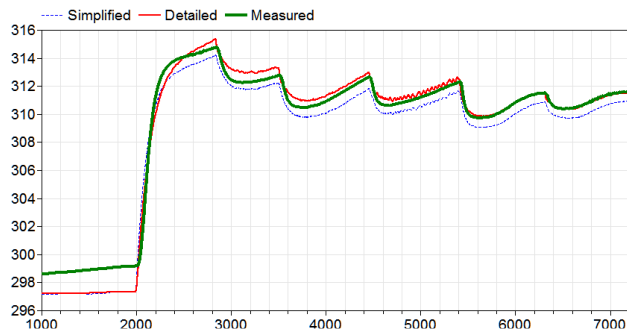


Figure 10. Case A outlet temperatures.

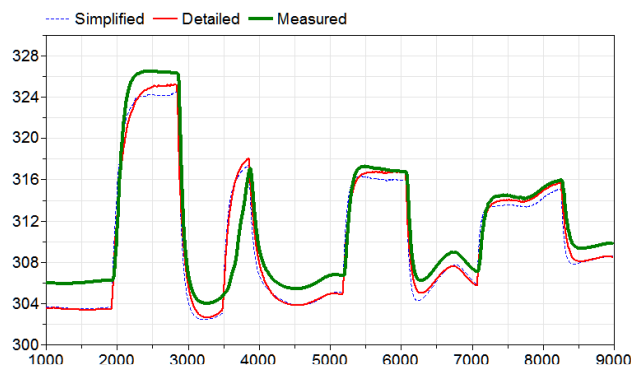


Figure 11. Case B outlet temperatures.

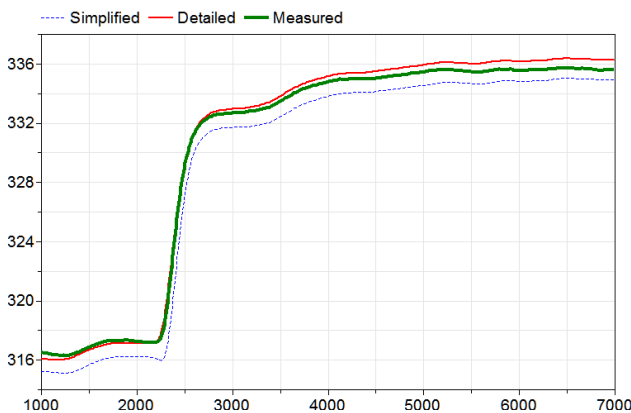


Figure 12. Case C outlet temperatures.

5 Discussion

For validation, as shown in Figure 10, Figure 11, and Figure 12, the collector outlet temperature values are very similar. An initial offset can be observed between the experimental data and the simulation models' results, especially for case A and B. This finding is observed because at the LER, when the experiment starts, the solar collector is covered, which avoids radiation entrance but also prevents direct convection phenomena to the ambient, while in the simulation models, this convection does occur (more losses, lower outlet temperature). Obviously, this offset, or a consequence of it, remains during the rest of the simulation.

Conversely, it must be noted that the outlet temperature slope with changing input values (mass flow rate, incident solar radiation, or inlet temperature) is very much alike in the three cases, which means that in both models the physical dynamics are properly modelled.

From these findings, it is concluded that both of the developed models are appropriate for representing the dynamic (and also the static) behaviour of a flat-plate solar collector. However, it is worth highlighting that depending on the application, one can be more suitable than the other.

Table 1. Models characteristics comparison.

	<i>Detailed</i>	<i>Simplified</i>
N. parameters	29	14
N inputs	9	5
N. continuous time states	7	3
CPU time ³	15.1	

As shown in Table 1, the Detailed Model has a larger number of parameters, whose values can be difficult to ascertain. It also has a greater number of time states involving a greater CPU time. However, due to the high degree of detail, this model is more suitable than the Simplified Model for analysing the influence of particular system variables, such as individual components' geometry or material, on collector performance. This capability is very valuable, for example, for modelling design improvements.

As for the Simplified Model, again paying attention to the results appearing in Table 1, it is ultimately more user-friendly. Simulation time is lower and therefore more suitable for control design purposes because it is normally necessary to perform numerous simulations to, for example, identify the controlled system. The only requirement is to have access to collector parameters obtained from the referenced standard test.

For cases in which direct access to standard test parameters is not possible, the Detailed Model may be the only way to obtain them. In this situation, the corresponding standard test load cases can be simulated using the Detailed Model instead of doing it experimentally, thereby obtaining the necessary Simplified Model parameters' values. This may also be a more affordable way.

Finally, when a controller is developed, a common final step in this process is the fine controller parameter tuning, which is normally carried out by driving the real system. However, simulations of the designed controller using the Detailed Model can be worked out instead, thus allowing for cost saving, and avoiding damage to the real system.

³Relation between Detailed/Simplified Models CPU times for Case A simulation.

6 Conclusions

Two flat-plate solar collector models have been developed in the Modelica language: the Detailed Model and the Simplified Model. It has been demonstrated that both provide a suitable representation of solar collector dynamic behaviour. However, there are differences between them that make them suitable for different applications.

The Detailed Model uses a high quantity of elements, parameters, and inputs, allowing for a complete and detailed analysis of the solar collector. This characteristic is suitable mainly for collector design purposes, such as studying components' material/geometry influence, etc.

The Simplified Model is a more workable model, with fewer but more accessible parameters based on standard tests. This type of model is primarily appropriate for control design purposes. In fact, within Ambassador Project, developed Simplified Model has been used to identify the flat-plate solar collector system via model simulations, getting to a transfer function (outlet temperature depending on inlet mass flow rate) consisting on a first-order system and a pure delay.

Acknowledgments

The research leading to the results presented in this paper (e.g., model development, analysis, and validation) has been developed under the project AMBASSADOR frame, which has received funding from the European Union Seventh Framework Programme [FP7/2007-2013] under grant agreement n°314175.

References

- R.W. Bliss Jr. The derivations of several "Plate-Efficiency Factors" useful in the design of flat-plate solar heat collectors. *Solar Energy* 3, 4, 55-64, 1959. doi: 10.1016/0038-092X(59)90006-4.
- J. Cadafalch. A detailed numerical model for flat plate solar thermal devices. *Solar Energy*, 83, 12, 2157-2164, 2009. doi: 10.1016/j.solener.2009.08.013.
- A. J. Chapman. *Fundamentals of Heat Transfer*, 1987.
- D. J. Close. A design approach for solar processes. *Solar Energy* 11, 2, 112-122, 1967. doi: 10.1016/0038-092X(67)90051-5.
- J. A. Duffie and W. A. Beckman. *Solar engineering of thermal processes*, 1991. doi: 10.1002/9781118671603.
- H. C. Hottel and A. Whillier. Evaluation of flat plate collector performance. *Transactions of the Conference on the use of solar energy II, Thermal Processes*, 74-1 04, 1955.
- H. C. Hottel and B.B. Woertz. The performance of flat-plate solar heat collectors. *Transactions of the ASME*, 64: 94-102, 1942.
- B. J. Huang, S.B. Wang. Identification of solar collector dynamics using physical model-based approach. *ASME. J. Dyn. Sys., Meas., Control*, 116(4):755-763, 1994. doi:10.1115/1.2899275.
- P. Isakson. Solar collector model for testing and simulation. *Final report for BFR project Nr. 900280-1, Building Services Engineering, Royal Institute of Technology, Stockholm*, 1995.
- W. Kamminga. The approximate temperatures within a flat-plate solar collector under transient conditions. *International Journal of Heat and Mass Transfer*, 28, 2, 433-440, 1985. doi: 10.1016/0017-9310(85)90076-6.
- S. A. Klein, J. A. Duffie, W. A. Beckman. Transient considerations of flat-Plate solar collectors. *ASME. J. Eng. Gas Turbines Power*, 96(2):109-113, 1974. doi:10.1115/1.3445757.
- S. López, I. del Hoyo. Proposal for standardization of Heat Transfer Modelling in NewThermal Library. *Proceedings of the 10th International Modelica Conference*, 2014. doi: 10.3384/ECP140961189.
- J. Muschaweck, W. Spirkl. Dynamic solar collector performance testing. *Solar Energy Materials and Solar Cells* 30, 2, 95-105, 1993. doi: 10.1016/0927-0248(93)90011-Q.
- A. Oliva, M. Costa, C.D. Perez Segarra. Numerical simulation of solar collectors: the effect of nonuniform and nonsteady state of boundary conditions. *Solar Energy*, 47, 5, 359-373, 1991. doi: 10.1016/0038-092X(91)90030-Z.
- E. Onillon. District energy flow optimization taking into account building flexibilities. *2nd Sustainable Places International Conference*, 2014.
- D. E. Prapas, B. Norton, et al. Response function for solar-energy collectors. *Solar Energy* 40, 4, 371-383, 1988. doi: 10.1016/0038-092X(88)90010-2.
- A. J. de Ron. Dynamic modelling and verification of a flat-solar collector. *Solar Energy* 24, 2, 117-128, 1980. Doi: 10.1016/0038-092X(80)90386-2.
- E. Sartori. Convection Coefficient Equations for Forced Air Flow over Flat Surfaces. *Solar Energy*, 80, 9, 1063-1071, 2006. doi: 10.1016/j.solener.2005.11.001.
- J. Schnieders. Comparison of the energy yield predictions of stationary and dynamic solar collector models and the models' accuracy in the description of a vacuum tube collector. *Solar Energy* 61, 3, 179-190, 1997. doi: 10.1016/S0038-092X(97)00036-4.
- Verein Deutscher Ingenieure. *VDI Wärmeatlas*. 1997.

Generic Modelica Framework for MultiBody Contacts and Discrete Element Method

Hilding Elmqvist¹, Axel Goteman^{1,2}, Vilhelm Roxling^{1,2}, Toheed Ghandriz¹

¹Dassault Systèmes, Lund, Sweden, {Hilding.Elmqvist, Toheed.Ghandriz}@3ds.com

²Lund Institute of Technology, Lund, Sweden, {axel.goteman, vilhelm.roxling}@gmail.com

Abstract

A generic framework for mechanical modeling of objects that collide and have contact is presented. It can be used in combination with the Modelica MultiBody library and to model granular objects using DEM (Discrete Element Method). The shapes of the objects are given by general triangular meshes.

Keywords: MultiBody models, Discrete Element Method, Collision detection, Contact handling

1 Introduction

Many real-world system behaviors depend on contact between mechanical bodies. Examples are walking, vehicle on a road, mechanisms in mechanical watches and many types of manufacturing machines.

1.1 Earlier Modelica Based Solutions

There have already been several developments to support collision handling in Modelica. In (Otter, et al 2005) is described a solution based on a collision handling software called Solid. The contact force calculations take into account the contact patch, i.e. also rotational friction torque is handled. The paper (Oestersötebier et al, 2014) introduces non-central contact blocks in which the contact surfaces are defined. (Hofmann, 2014) discusses the use of the Bullet Physics Library and deepest point penetration for force calculations. Unfortunately this point might not be unique which then results in unrealistic simulation results.

1.2 Discrete Element Method

The Discrete Element Method (DEM) has a focus on handling many interacting objects. Typically, both positional and rotational degrees of freedom are handled, i.e. 6 DOFs per object. The geometries can be complex, e.g. polyhedral. Many different force models can be used depending on what matter is studied.

It has been noted that simply using penetration depth, which is often calculated by collision packages, are not suited for continuous time simulation because of the discontinuity in the forces and where the forces act. DEM typically handles this by calculating penetration volume. See, for example, (Chen, 2012).

(Hippman, 2003, 2013) also considers the penetration volume and makes force calculations based on the surface patches.

1.3 Contribution

A generic framework for mechanical modeling of objects that collide and have contact is presented. It can be used in combination with the Modelica MultiBody library and to model granular objects using DEM (Discrete Element Method). The shapes of the objects are given by general triangular meshes. The special case of sphere is also supported in order to handle tens thousands of objects for DEM. The contact handling is organized using ExternalObjects, i.e. C and C++ code. Each body in the scene registers its current position which is given as the solution of the Modelica motion equations. After that a centralized routine of the scene calculates and adds all forces between pairs of bodies in contact. The force calculation is done using the intersection volume found by the CSG (Constructive Solid Geometry) intersect operator. We have used a generalization of the Hertz contact model developed by (Nassauer, 2013), where the force is proportional to \sqrt{Vd} , with V =penetration volume and d =penetration depth. The force is acting in the centroid of the penetration volume. The details are given in section 6.1.

1.4 Example: Simple Objects

As an example of how the presented library can be used, consider the billiard like set-up in Figure 1 of 100 layers of balls, i.e. 5050 balls on a table plus one ball hitting from below.



Figure 1. 5050 balls hit from below

The area around where the first collision happens is enlarged in Figure 2.

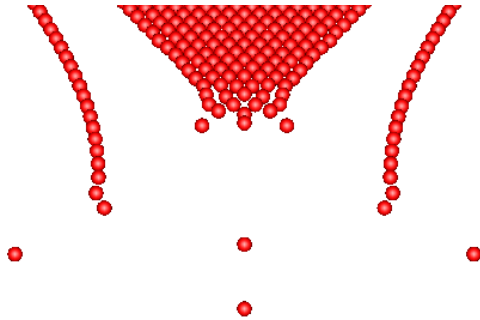


Figure 2. Zoomed in on the first colliding balls
Modeling such scenario is quite simple:

```
model Billiard
parameter Integer layers=100;
parameter Integer n=div(layers*(layers+1),2);
inner CollidingWorld collidingWorld;
Sphere sphere[n]
(x0={{layer(i)*sqrt(3)/2,
column(i)-(layer(i)-1)*0.5, 0}
for i in 1:n}, each radius=0.5) ;
Sphere sphere1(x0={-5,0,0}, v0={1,0,0}, radius=0.5);
end Billiard;
```

The trivial functions layer and column calculates the position of the i^{th} ball in the triangle.

2 Modeling Solids

2.1 Triangle Mesh

One popular method of representing solids is to define its closed surface by a set of triangles with its counter clockwise normal pointing outwards. The following Modelica record is used:

```
record TriangleMesh "Defines solid by triangle mesh of its surface"
parameter Real vertices[: ,3] "3D coordinates of points";
parameter Integer triangles[: ,3](min=1)
"Triangle structure based on vertices (index array)";
end TriangleMesh;
```

i.e. all vertices are defined in a separate vector and the triangles are defined using indices into this vector.

2.2 Solids Defined by Functions

A cube can be defined as a function:

```
function cube
output TriangleMesh mesh=TriangleMesh(
vertices={{0,0,0}, {0,1,0}, {0,0,1}, {0,1,1},
{1,0,0}, {1,1,0}, {1,0,1}, {1,1,1}},
triangles={{1,4,2}, {1,3,4}, {1,5,7}, {1,7,3}, {5,8,7}, {5,6,8},
{2,4,8}, {2,8,6}, {4,3,8}, {3,7,8}, {1,6,5}, {1,2,6}});
end cube;
```

The library contains a set of functions for defining the triangular meshes of basic primitive shapes such as box, cylinder and sphere. In addition, functions for translating, rotating and scaling triangle meshes are available.

2.3 Polygon Extrusion

A convenient method of defining solids, especially for early concept studies, is to extrude polygons.

2.3.1 Editing Polygon as Icon

Polygon editing is available in Modelica tools since one of the icon primitives is Polygon. Dymola has an API function to retrieve annotations from Modelica classes. This function is used to define a parameter vector of 2D coordinates. We propose that such a function is included in the Modelica standard.

An example of such use is shown in Figure 3 for a pallet level of a watch. A picture was imported into the icon editor and a polygon was drawn over the picture.

2.3.2 Concave Polygon Triangulation

The polygon, which can be concave, needs to be triangulated in order to fit into the triangular mesh representation.

It is assumed that the polygon is defined counter clockwise. The vertices of the polygon need to be traversed and inspected for “ears”, i.e. when the edges make a “right turn”. A simple algorithm is to remove vertices at “left turns” and generate the corresponding triangles first. The polygon is therefore traversed as long as there are at least 3 remaining vertices.

2.3.3 Extrusion

The extrude function takes a potentially concave polygon as input and an extrusion height parameter. It triangulates the polygon and converts to 3D coordinates for front and back. The extrusion sides are trivially triangulated by splitting the rectangles defined by the polygon vertices from the front and back polygon respectively.

The resulting triangular mesh for extruding the pallet lever polygon is shown in Figure 3. The triangle vertices are colored red, green and blue and interpolation has been performed over the triangle surface.

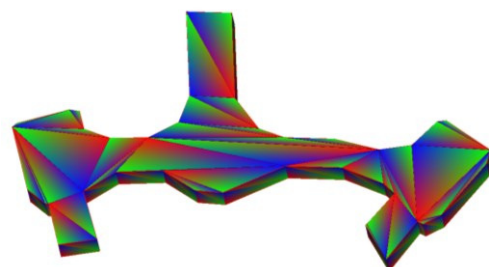


Figure 3. Triangle mesh of extruded part

2.3.4 Regular Forms

For regular forms such as a gear wheel, it's convenient to just define the polygon for one tooth and use replication to define the entire form. In order to replicate around a circle, the polygon needs to be rotated. Functions to translate, rotate and scale polygons are defined.

Figure 4 shows how the above operations can be used to define an escape wheel of a watch.

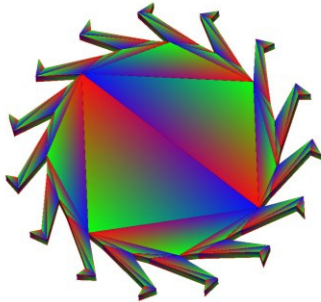


Figure 4. Triangle mesh of extruded regular form

2.4 Constructive Solid Geometry Operations

More complex geometries can be defined using constructive solid geometry (CSG). It enables combining triangular meshes by the operations: union, difference and intersection.

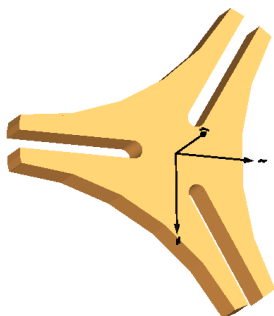


Figure 5. Triangle mesh of part created by CSG operations

The solid in Figure 5 has been created using the following Modelica code (the parameters are defined in section 8.2):

```
mesh := rotate(Cylinder(radius, height=height), {pi/2,0,0});
for i in 1:nSlots loop
  mesh := difference(mesh, rotate(translate(
    Box({slotLength,height,slotWidth}),
    {radius-slotLength,-height,-slotWidth/2}),
    {0,2*pi/nSlots*(0.5+i),0}));
  mesh := difference(mesh, rotate(translate(rotate(
    Cylinder(slotWidth/2, height=height), {pi/2,0,0}),
    {radius-slotLength,0,0}), {0,2*pi/nSlots*(0.5+i),0}));
  mesh := difference(mesh, rotate(translate(rotate(
    Cylinder(stopArcRadius, height=height), {pi/2,0,0}),
    {centerDistance,0,0}), {0,2*pi/nSlots*i,0}));
end for;
```

A Cylinder is first constructed. A for loop is then used to remove boxes and cylinders from the solid by using the CSG difference operator.

This method of defining solids, i.e. a textual definition of object creation, extrusion, CSG operations, etc, can also be found in certain CAD packages such as OpenSCAD and OpenJSCAD.org. The latter uses JavaScript to define the solid objects and the operations.

2.5 Use of CAD data

The presented Modelica library also contains functions for reading triangle meshes from CAD files. The current implementation allows reading DXF files. By using converters other CAD formats can also be used such as VRML.

3 Modeling Idiom for Pairwise Coupling of Objects

It is important that the end user does not have to care about setting-up individual possible contact pairs. It should be possible to just drag an object into the scene, which is possible in Playmola (Elmqvist, et al., 2015), and automatically get collision behavior.

A unique Integer id needs to be defined by the user both for the solutions of (Otter, et al., 2005) and (Oestersötebier et al., 2014).

Our solution solves this problem as follows. The contact handling is organized using ExternalObjects, i.e. C and C++ code. Each body in the scene registers its position which is given as the solution of the Modelica motion equations. After that a centralized routine of the scene calculates and adds all forces between pairs of bodies in contact. These forces are then retrieved for each body and used in their motion equations.

The difficulty is to properly synchronize the external calculations of the forces. All current positions and orientations must be known then.

We use inner outer constructs together with flow variable declarations in the following way. When a body calls `setBodyTransformation`, it also assigns `bodyMoved` which is defined as `sync.synchronize.done`. When the body calls `getBodyForces` an extra term is added: `collidingWorld.forceCalculated`.

```
model Body
  extends Contacts.Synchronize.CollidingObject;
  Real bodyMoved = sync.synchronize.done;
  Modelica.SIunits.Force f[3];
equation
  bodyMoved = setBodyTransformation(...);
  f = getBodyForces(b,time) +
    collidingWorld.forceCalculated;
  ...
end Body;
```

The term `forceCalculated` will not have a value until all bodies have defined `bodyMoved` and the `calculateForces` function of the external scene object has been called. This is accomplished in an algorithm in the inner instance of `CollidingWorld`:

```

model CollidingWorld
  ExternalScene scene=ExternalScene(id="s");
  Synchronize.SynchronizeConnector synchronize;
  Real bodyMoved = synchronize.done;
  Real forceCalculated[3];
  Real dummy;
equation
  synchronize.do = 0;
algorithm
  dummy := bodyMoved;
  calculateForces(scene);
  forceCalculated :=fill(0, 3);
end CollidingWorld;

```

The trick is then that `bodyMoved` of `CollidingWorld` should depend on the `bodyMoved` of all the bodies. `bodyMoved` is defined using a Real flow variable `done`.

```

connector SynchronizeConnector
  Real do;
  flow Real done;
end SynchronizeConnector;

```

This means that all the outer `done` variables of `CollidingWorld.synchronize` are summed together with the inner `done` of `CollidingWorld.synchronize`. However, to achieve this zero sum semantics, there needs to be some connections to `collidingWorld.synchronize`. Each body extends `CollidingObject` and gets the outer declaration, an instance of `SynchronizeModel` and the needed connect statement.

```

model CollidingObject
  outer CollidingWorld collidingWorld;
  SynchronizeModel sync;
equation
  connect(collidingWorld.synchronize, sync.synchronize);
end CollidingObject;

model SynchronizeModel
  SynchronizeConnector synchronize;
end SynchronizeModel;

```

This is indeed complex. However, a useful consequence of Modelica semantics of inner-outer combined with connectors having a flow variable. Fortunately, the end user does not need to care about this.

This idiom is documented here since it might be useful in other circumstances when external objects are involved and careful synchronization of calls to external functions are necessary.

4 Extension to MSL MultiBody library

A general body with collision behavior and general triangle mesh shape has been developed as a wrapper to `Modelica.Mechanics.MultiBody.Parts.Body`. The mass and inertia are automatically calculated. It has the usual frame connector, i.e. it can be used together with joints and force elements in the usual way.

5 Contact detection

Often, the most time consuming part of a DEM simulation is the contact detection. It is therefore crucial that it is as efficient as possible, for speed considerations. The contact detection can be divided into two phases, the broad phase and the narrow phase. The broad phase consists of finding potential collision pairs among all the bodies in the scene, while in the narrow phase, those pairs are checked in detail for collision. For more detailed descriptions than present in this section, see (Goteman, et al., 2015).

5.1 Without a broad phase

To begin with, a broad phase is not necessary in cases where the number of bodies in the scene, n , is low. In those cases the body shapes may also be more complex, which means that the narrow phase will take the majority of the time, and a broad phase would not make an impact anyway.

Without a broad phase, every body has to be checked against all other bodies, resulting in a time complexity of $O(n^2)$. This does not mean that a detailed investigation is needed between every pair of bodies. Normally, a *bounding volume* of every body is determined. If the bounding volumes of two bodies intersect, that pair can be checked in more detail. The most common bounding volume type is the *axis aligned bounding box (AABB)*, but one can also use e.g. the *bounding sphere* (centered in the centroid of the objects).

5.2 Broad phase

As n increases, it gets expensive to even check bounding volumes for all bodies. The point of having a broad phase is to remove the quadratic time dependency. The most common approaches uses some kind of tree structure into which the objects are placed, resulting in an $O(n \log(n))$ time complexity.

As an example, imagine 1000 balls placed in a straight line, every pair being precisely at contact. Without a broad phase, approximately half a million bounding volume checks would be carried out, assuming that every pair of bodies is only checked once. With a good broad phase however, only 999 checks would be needed.

objects generate eight intervals, minimizing the gap removal.

Step three could not be effectively parallelized, since no effective method was found for determining the number of generated collision pairs per interval. Without a guaranteed upper limit on collision pairs, the Morton encoding method could not be used, and using atomic operations proved to be too slow, due to the high number of access conflicts.

5.3 Narrow phase

When the AABBs of two bodies are colliding, a method is needed to detect if the bodies are actually colliding. If a collision is detected, the overlapping region also has to be determined in order to calculate the resulting forces.

For this, the CSG (Constructive Solid Geometry) intersect operator, using BSP (Binary Space Partitioning) trees, is used (Segura, 2013). It is a suitable method since it works well and fast on arbitrarily shaped bodies that are described by their polygon surfaces.

In a BSP tree, every node has two children. A node has a plane in 3d space, and its left child represents one of the half spaces created by this plane, and the right child represents the other. All operations on the tree are naturally defined recursively. BSP trees are widely used in e.g. computer graphics, where the planes might be chosen as the walls of a room.

When using BSP trees for CSG operations, the planes are chosen as coplanar with the surface polygons of the bodies. It results in the left child being inside the polygon, and the right child being outside (or the other way around, depending on how you define your tree), see Figure 8.

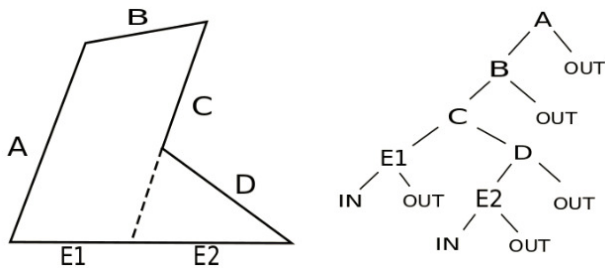


Figure 8. BSP tree

Note that polygon (line in 2D) E has to be split by the plane coplanar to polygon C in the tree construction, but not A by D. This is because the tree structure depends on which polygon is chosen as root.

It is now possible to efficiently determine e.g. which part of a body surface is inside another. Polygon by polygon, the surface can be traced down the tree, until it comes to an empty node, which is either in or out. The polygons are split if they span both sides of a plane.

The implementation is made in C++, building on principles from a ported version of the JavaScript library csg.js, by (Wallace, 2012).

5.3.1 Multiple Contacts

The intersection from a collision between concave bodies may consist of multiple contact regions. The result of the CSG intersection contains no information about this, so an algorithm is needed, that splits up the provided set of polygons (if needed). A short description of a solution:

The algorithm assumes that the original intersection consists of one or more closed volumes, which are to be represented as a vector of intersections. The algorithm goes through all polygons in the original intersection and checks the polygons vertices against the vertices of each intersection. If no match is found, a new intersection is created. If one match is found, the polygon is added to that intersection. If multiple matches are found, the corresponding intersections are merged and the polygon added.

5.4 Iterative reformulation

Motivated by the fact that the narrow phase generally was the computationally heaviest part of a simulation, except for simulations of spheres, attempts to accelerate this phase were made. The critical part of the narrow phase is when polygons of one body are traversing the other body's tree, so that became the focus.

Every polygon of a body traverses the tree independently, which motivated to do the traversal in parallel. Then, by traversing all polygons in all collision pairs simultaneously, high parallelism is possible both with few complex objects, and many simple ones, see figure 9.

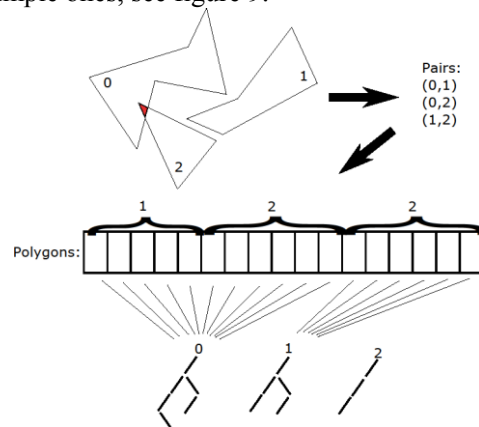


Figure 9. Illustration of the new traversal algorithm.

To achieve this kind of traversal, a major restructuring of the CSG algorithm were carried out. Incidentally, this restructuring resulted in a large performance boost for CPU execution as well. In fact, it performed as good as or better than the GPU in most cases.

The three major reasons for the lack of GPU acceleration are:

- High instruction divergence. Nearby threads may take different paths in the tree.
- High data divergence. Nearby threads might access non-coalesced memory locations.
- Each thread requires a large amount of registers, decreasing the occupancy.

6 Force Calculations

6.1 Normal force

When the intersection between two bodies is determined, the resulting forces need to be calculated. Because the colliding bodies are arbitrarily shaped, the overlapping region (the intersection) is given as a polyhedron with arbitrary shape, which implies that a classification of type of collision is hard to make. Thus a collision type independent model is needed. One such model is proposed in (Nassauer, 2013), where the contact force for elastic response is volume dependent. The derivations are based on Hertz model for contacts between spheres. With the assumption that the contact region is small with respect to the bodies in contact, it leads to:

$$\mathbf{F} = E k \sqrt{V d}$$

Here E is Young's modulus of the objects, V is the volume of the overlapping region, d is the penetration depth and

$$k = \frac{4}{3\sqrt{\pi}}$$

It is shown that this is actually a generalization of Hertz model.

The force is applied at the centroid of the overlapping region. To determine the force direction, constant pressure is assumed within the intersection. The direction can then be determined by weighing each polygon's normal with its area, sum over the polygons from one of the bodies, and normalize:

$$\mathbf{n}_f = \frac{\sum_j A_j \mathbf{n}_j}{\sum_j A_j}$$

A_i and \mathbf{n}_i are the area and normal direction of polygon i respectively. This gives a behavior similar to that of a body floating in a liquid, the only difference being the assumption of constant pressure. An illustration of the polygonal contributions for the force direction is shown in Figure 9, taken from the debug window developed together with this package.

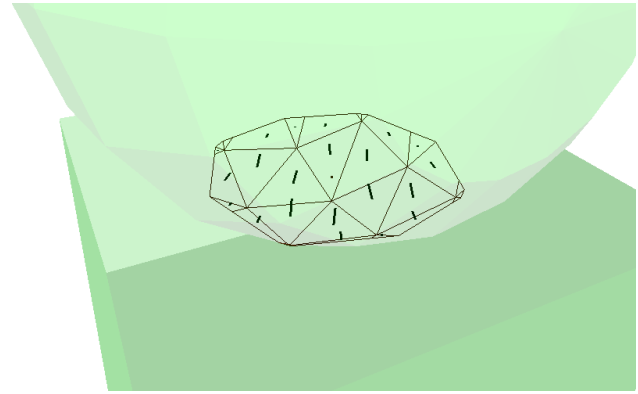


Figure 9. Intersection volume and forces

The penetration depth is defined as the extension of the overlapping region in the direction of \mathbf{n}_f , see Figure 10, and can be found as

$$d = \max_j (\mathbf{n}_f \cdot \mathbf{p}_j) - \min_j (\mathbf{n}_f \cdot \mathbf{p}_j).$$

\mathbf{p}_i is the position of vertex i , and j ranges over all vertices in the intersection.

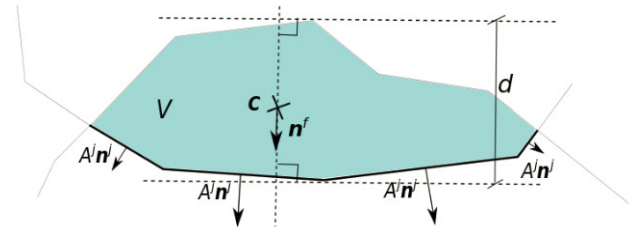


Figure 10. Penetration depth

6.2 Additional forces

In this context, the normal force is not sufficient to describe the contact interaction between bodies. (Nassauer, 2013) also proposes models for damping and friction, described below.

6.2.1 Damping

Energy dissipation can be introduced by modifying the normal force equation to

$$\mathbf{F} = E k \sqrt{V d} (\mathbf{1} + c \mathbf{v}_d),$$

where c is the damping constant and \mathbf{v}_d is the relative velocity between the bodies in the orientation of \mathbf{n}_f . The points for which the velocities are calculated are chosen as the centroid of the intersection for both bodies.

6.2.2 Friction

The classical Coulomb friction model turns out to be very hard to implement in DEM simulations. There are many reasons for this, the most important being that, in the case of static friction, all other forces acting on the body have to be known.

So instead, the following, velocity dependent model is used:

$$F_f = \left((2\mu_{s*} - \mu_k) \frac{x^2}{x^4 + 1} + \mu_k - \frac{\mu_k}{x^2 + 1} \right) F$$

F is the normal force, $x=v_t/v_s$, where v_t is the magnitude of the relative tangential velocity between the bodies, and v_s is the velocity for transition from static to kinetic friction. v_t can be found by projecting the relative velocity between the bodies, again taken at the centroid of the intersection, onto the plane of which \mathbf{n}_f is normal. Also

$$\mu_{s*} = \mu_s (1 - 0.09 \left(\frac{\mu_k}{\mu_s} \right)^4),$$

where μ_s and μ_k are the coefficients of friction for static and kinetic friction respectively. This generates the following friction behavior.

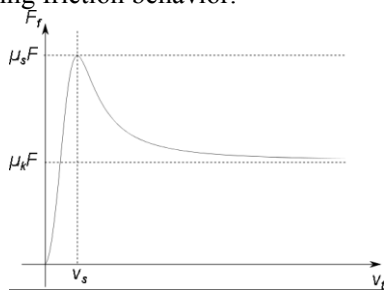


Figure 11. The friction model

6.2.3 Rotational damping

To simulate the friction caused by rotation at the contact point, a model with a damping effect is proposed. It applies a torque in the opposite direction of the relative angular velocity at the contact point (in the direction of \mathbf{n}_f). Its magnitude is proportional to the angular velocity, the area of the contact region, and the normal force, according to

$$\mathbf{M}_r = -AFc_r(\boldsymbol{\omega}_1 - \boldsymbol{\omega}_2)\mathbf{n}_f$$

A is the area of the contact region, c_r is a constant for calibration of the model, and $\boldsymbol{\omega}_i$ is the angular velocity of body i , which is the same in any point of a rigid body. The area can be approximated by projecting the triangles from one body onto the plane to which \mathbf{n}_f is normal, and summarize:

$$A = \sum_j A_j \mathbf{n}_j \mathbf{n}_f$$

This model does not take into account the fact that the friction force most probably is not uniformly distributed across the contact area. A better solution (not yet implemented) would be to integrate over the area, and apply the friction model from section 6.2.2 on the triangles. F should then be replaced by the corresponding local contribution to the normal force, and v_t by the tangential component of $\boldsymbol{\omega} \times \mathbf{r}_i$, where \mathbf{r}_i is the position of the triangle relative to some fix point, e.g. the centroid of the intersection.

6.3 Additional algorithms for polyhedrons

6.3.1 Volume and centroid

To complete the force calculations, algorithms to compute the volume and the centroid of the intersection are needed. Those algorithms are also needed to compute center of mass and mass of a triangular meshed body. This is done by (Nürnberg, 2013), for closed polyhedrons of arbitrary shape. Conveniently, the algorithms only depends on the surface of the polyhedron, which is what is produced from the CSG intersection operation.

6.3.2 Moment of inertia

An algorithm to compute the inertia tensor of the bodies is also needed. This can be done for an arbitrary polyhedron (with triangle faces) in the following way:

Pick a point, e.g. the point around which the inertia tensor is needed. If another point is chosen, a simple translation to the desired point has to be carried out afterwards.

The integration over the body's volume can be transformed to a sum of integrals over tetrahedrons. Those tetrahedrons are the ones created by connecting the chosen point with the triangles of the body surface. Now, for one such tetrahedron, if the normal of the triangle from the object's surface points into the tetrahedron, the contribution from this tetrahedron is to be subtracted, otherwise added, to the total inertia tensor.

What remains is to actually calculate the moment of inertia of arbitrary shaped tetrahedrons. (Tonon, 2004) presents expressions for this in terms of the vertex coordinates.

7 Animation

7.1 Additional debug window

In order to allow for debugging and analysis of contact behavior, OpenGL was used to create a new animation window for Dymola. This window has expanded capabilities in order to simplify debugging of contact mechanics.

It allows viewing of the individual polygons of a body, which gives the option of viewing the intersections in the event of a collision. The window also supports drawing of the forces, both the contribution of every individual intersection polygon and the combined collision force. The resulting torque of the collision force is also viewable.

7.2 New ModelicaServices Model for Triangle Mesh

The ModelicaServices package which might have different implementations for different tools, has some models for animation of certain predefined shapes such

as box, cylinder and sphere, DXF-files and parametric surfaces. Parametric surfaces are defined by a function of two independent parameters and return the corresponding 3D position.

The above capabilities are not sufficient for animation of triangular meshes. We propose that ModelicaServices is extended with such a model. It will be similar to ModelicaServices.Animation.Surface, defining the surface by:

```
input Real vertices[:,3]
  "3D coordinates of points";
parameter Integer triangles[:,3]
  "Triangle structure based on vertices"
```

Note that vertices can be time dependent, i.e. it is possible to animate flexible bodies.

Three different coloring schemes are proposed: one color for entire shape, separate color for each triangle and separate color for each vertex with interpolation over triangle. The last most powerful alternative supports animating stresses in flexible bodies.

8 Examples

8.1 Lever Escapement

The lever escapement was invented around 1755 and is used in watches to convert an oscillation to a rotational motion, see Wikipedia ("Lever escapement"), Figure 12:

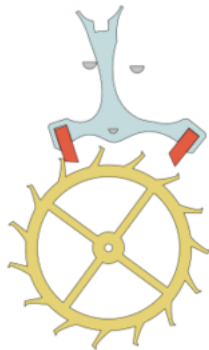


Figure 12. Lever escapement of mechanical watch

There are two contact problems: One for making sure the rotation ticks at the desired frequency; the other to insert more energy from the spring of the watch to the oscillator.

The shape of the pallet lever was extracted from (British Horological Institute, 2011) and a polygon was created and extruded. The escape wheel was defined as a regular form as described in section 2.3, see Figure 13.

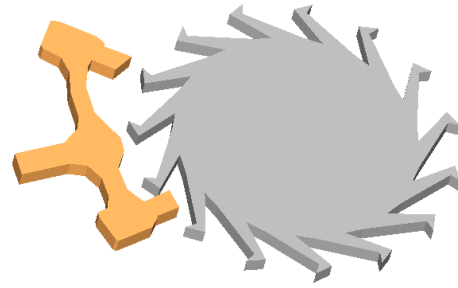


Figure 13. Pallet lever and escape wheel

8.2 Geneva Mechanism

The Geneva mechanism has been used since long ago to achieve intermittent motion (Bickford, 1972). It is used, for example, in mechanical watches and in film projectors to move the film.

The drive wheel (grey in the figures below) has a pin which rotates the output wheel (yellow) when in contact in the slots. When the pin is not in contact, the output wheel locks its locking surface by sliding against the locking ring of the drive wheel.

The dynamics is thus quite complex and there is a fast change in the acceleration when the drive pin enters and leaves the slot.

8.2.1 Parametric mechanism

Depending on how many output slots there are, the geometry of the parts is given. The Modelica function generating the output wheel by CSG operations in section 2.4 has the following variable declarations showing the dependencies.

```
input Real radius=1 "Geneva wheel radius";
input Integer nSlots(min=3)=3 "Number of driven slots";
input Real pinDiameter=radius/10 "Drive pin diameter";
input Real clearance=radius/100;
input Real height=radius/5;
output TriangleMesh mesh;
protected
Real pi=Modelica.Constants.pi;
Real centerDistance = radius/cos(pi/nSlots);
Real driveCrankRadius = sqrt(centerDistance^2-radius^2);
Real slotLength = radius + driveCrankRadius - centerDistance;
Real slotWidth = pinDiameter + clearance;
Real stopArcRadius = driveCrankRadius - pinDiameter*1.5;
Real stopDiscRadius = stopArcRadius-clearance;
Real clearanceArc = radius*stopDiscRadius/driveCrankRadius;
```

It is therefore possible to define a parametric mechanism, i.e. that the shapes of all parts of the mechanism are parametric. The Geneva mechanism with $nSlots=3$ is shown in Figure 14.

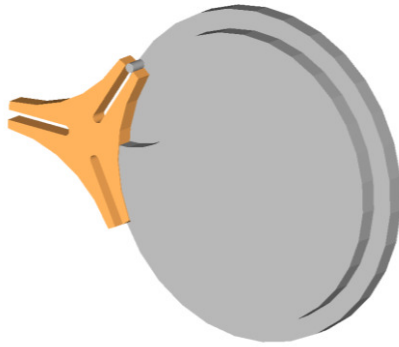


Figure 14. Geneva mechanism with 3 slots

The Geneva mechanism with $nSlots=6$ is shown in Figure 15. Notice that the drive wheel is then relatively smaller.

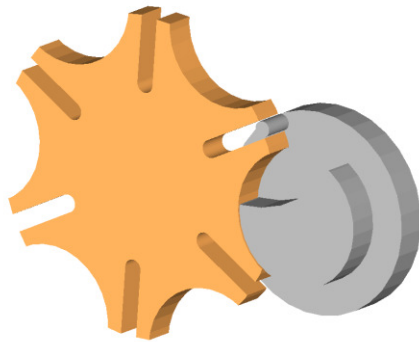


Figure 15. Geneva mechanism with 6 slots

The angle of the output wheel is plotted in Figure 16 in the case of $nSlots=6$:

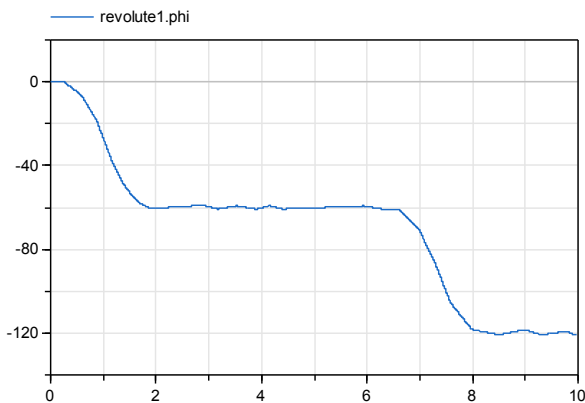


Figure 16. Output angle when 6 slots

It should be noted that the contact handling is very complex. In particular, the locking surface is concave. It is sliding against the convex locking ring which has smaller radius. So theoretically there would be one contact surface. However, due to the discretization of

triangulated mesh, there might be several simultaneous contact surfaces which our approach handles.

8.3 Bucket Digging in Pile of Belgian Blocks

Some of the roads in the old central parts of the city of Lund, Sweden are built by Belgian Block stones, see Figure 17.



Figure 17. Belgian Block stones

To demonstrate that our solution can handle many objects (DEM) defined by triangular meshes, we want to calculate the force needed on the bucket to grab the stones by an excavator or loader tractor, see Figure 18. The time to simulate 4 seconds of real time took 27 minutes. During that time more than 7 million collisions took place.

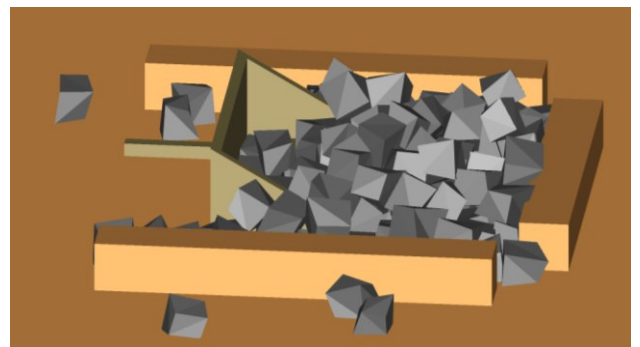


Figure 18. Bucket digging pile of Belgian Block stones

The plot of the force is shown in Figure 19 when the bucket has constant velocity.

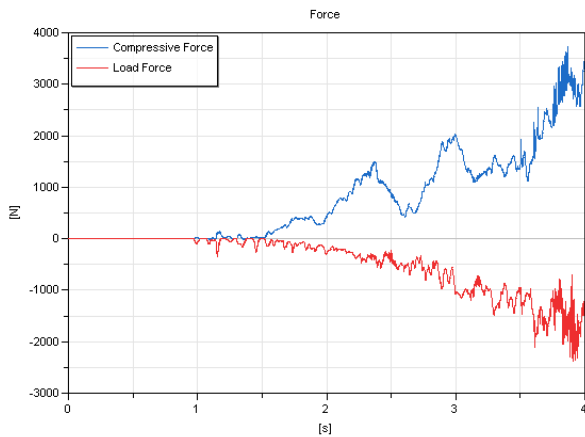


Figure 19. Plot of horizontal and vertical forces during digging

8.4 Tippe Top

The Tippe Top is a special toy that has fascinated a lot of physicists for a long time. It consists of a hollow sphere with a stem, as can be seen in Figure 20.



Figure 20: Snapshot of Tippe Top inversion, from left to right

As a result of its special geometry, its center of mass is located below the center of the sphere. When spun, the Tippe Top will invert itself, and spin on the stem, Figure 20. Since the behavior of the Tippe Top is quite complex, the fact that we can simulate it serves as a good indicator of the potential of this package.

Simulating the behavior until inverting after 3.3 seconds took 12 minutes.

9 Extensions to Flexible Bodies

The presented framework could naturally be extended to flexible bodies as well. This section introduces related work not yet integrated in the framework which shows the possibilities.

A prototype library has been developed in Modelica to model and simulate *planar* flexible multibody systems which also has the ability of simulating contact problems. The simulation of the flexible multibody system is based on the floating frame of reference approach and a model reducing technique, Kraig-Bampton method (Ghandriz, 2014, Shabana, 2013 and Simeon, 2013). The geometry of a body is a set of polygons defining the outer and inner boundaries of the body. A standalone code was developed for generating the finite element mesh by implementing Delaunay triangulation (Shewchuk, 2012). The reduced model is generated as Modelica code. Once a flexible or a rigid

body is generated inside Dymola it can be used together with joints, drivers, constraints, etc. to build a multibody system using the PlanarMultiBody library (Zimmer, 2012).

Having solved the equation of motion of the model, the nodal elastic deformations of the flexible components are retained from the modal coordinates which is used to calculate the time history of the planar stresses.

A few examples will be given below to show the capabilities.

9.1 Flexible Bodies – FEM

The first example is a simplified version of a mechanism so called *wing variable camber leading edge flap* used in Boeing aircraft (Cole, 1967). A similar mechanism is shown in Figure 21.



Figure 21. Wing variable camber leading edge flap

The behavior of the system can be analyzed using the planar flexible library. The simplified mechanism in Figure 22 consists of 13 rigid and two flexible bodies.

The bending of the flap and the resulting stress distribution at an instance of time can be seen in Figure 23.

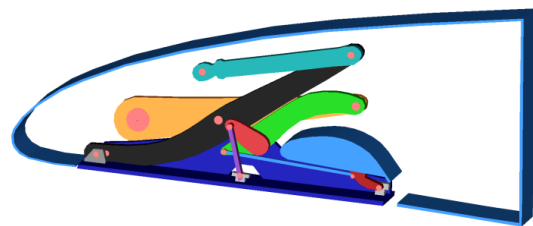


Figure 22. Folded wing flap

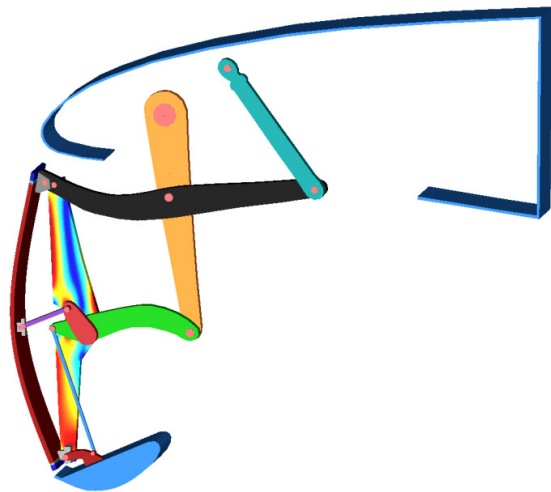


Figure 23. Unfolding wing flap

9.2 Flexibility and Topology Optimization

One of the excellent applications of flexible multibody dynamics can be realized as its combination with the theory of structural optimization. In particular, structural topology optimization is a part of conceptual design of a mechanical product where the material distribution, i.e. topology, of the body is iteratively updated to reach a constraint optimal state (Bendsoe, 2003). It means that, for example, with the optimal topology, the body can be stiffer or stronger but lighter. It is the purpose of many mechanical design engineers to build a mechanical part which shows the highest strength on the operation with the minimum amount of material used. In (Ghandriz, 2015) a method for applying structural topology optimization on multibody systems (TOMBS) can be found; where, large rotational and transitional motion, transient inertia and reaction forces of flexible bodies are accounted for in the optimization process.

For applying TOMBS on a flexible multibody system it is required to solve equation of motion in every optimization iteration; thus, the modal reduction and retaining all nodal elastic deformations must also be repeated in accordance with the new topology.

We apply TOMBS on one of the flexible bodies in the above example. The optimization problems are defined as to minimize the sum of the strain energy stored in the body over the operation time, while the maximum allowed volume is 60% of the initial volume shown in Figure 22.

If the thickness of the non-optimized body is changed such that it has the same weight as the optimized body, the generated stresses during operation in the body with optimal topology is smaller than the stresses of the non-optimized one. Figure 24 shows the change of the maximum stress of the two designs (optimized and non-optimized with reduced thickness) over time. The stress distribution of the non-optimized body with reduced thickness is shown in Figure 25 (upper part). The optimal result is shown in

Figure 25 (lower part). The colors are set for illustrative purpose. The highest stress is well below the yield stress. In later design stages, the high local stresses can be cured using shape optimization.

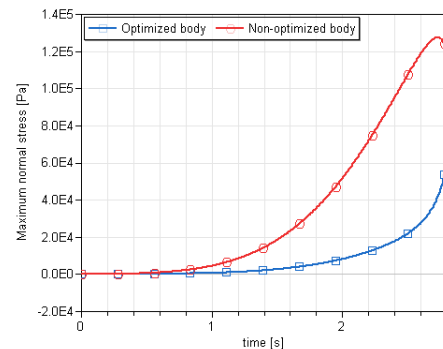


Figure 24. Maximum stresses for optimized and non-optimized body

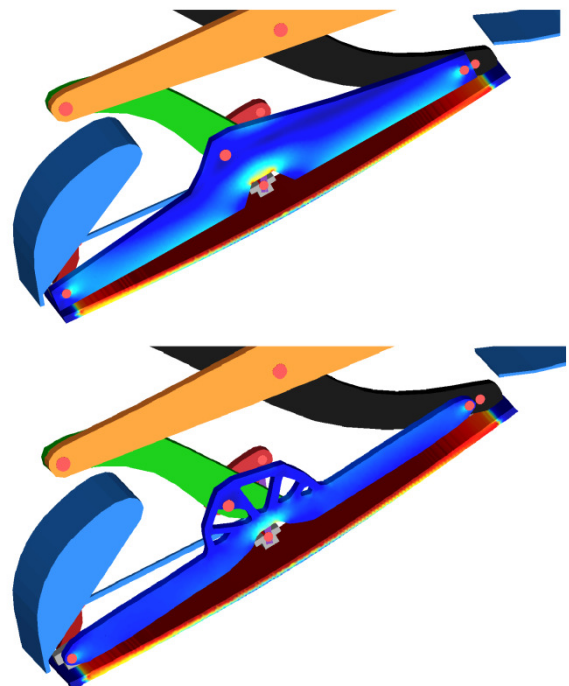


Figure 25. Stress distribution of the non-optimized body with reduced thickness (upper part); resulting shape of topology optimization (lower figure)

9.3 Flexibility and Contact

The last example is the lever escapement where both bodies are flexible. In flexible bodies, the forces generated due to the contacts must be distributed to the nodes of the finite elements involved in contact. The portion of the total contact force which each node receives is proportional to the node's penetration depth and its distance from the center of the contact region.

To obtain a more exact local deformation of the finite elements involved in contact, corresponding

nodes must be excluded from the modal reduction, i.e. they should be in the set of Master nodes in Kriag-Bampton method. Figure 26 shows the finite element mesh. In Figure 27, three snapshots of the model and the resulting stresses at the moment of contact are illustrated.

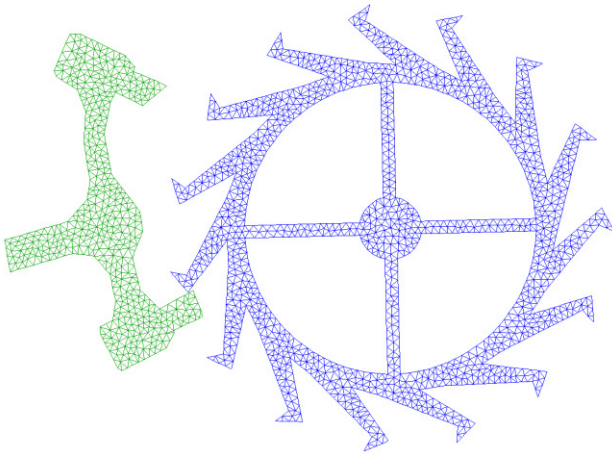


Figure 26. FEM mesh

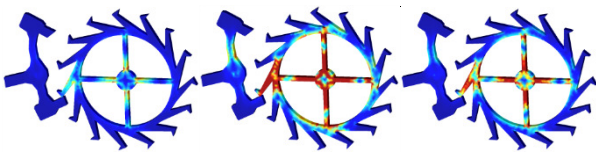


Figure 27. Stresses during the contact

It is interesting to see that the highest stresses in the Pallet body, i.e. the body on the left, are caused by the inertia forces. These stresses cannot be captured if the problem is analyzed statically.

10 Conclusions

A new Modelica framework for collision handling and DEM has been presented. It allows construction of 3D parts by use of Modelica functions for CSG in the early concept phase. CSG is also used to find contact region. Special considerations are taken in order to be able to handle DEM.

A discussion about the possibility to extend this framework to FEM with contact and for topology optimization is given.

We propose that this framework serves as a starting point for a working group within Modelica Association to define a standard Modelica library for collision, contact and DEM.

Acknowledgements

This work has partly been performed as a master thesis project at Lund Institute of Technology. The first author served as an industrial advisor and Michael Doggett as the formal supervisor.

The authors want to thank Hans Olsson for extending Dymola with the capability to animate triangular meshes.

References

- Bendsoe M. P., Sigmund O. (2003): Topology optimization: theory, methods and applications. Springer Science & Business Media
- Bickford J.H. (1972). Geneva Mechanisms. Mechanisms for intermittent motion. New York: Industrial Press inc. 128. ISBN 0-8311-1091-0, http://ebooks.library.cornell.edu/k/kmoddl/pdf/002_010.pdf
- British Horological Institute (2011): Drawing Clock and Watch Escapements - Distance Learning Course. pp 17-30, <http://www.bhi.co.uk/sites/default/files/Drawing%20Escapements%20Version%201.4%20DS.pdf>
- Chen J. (2012): Discrete Element Method for 3D Simulations of Mechanical Systems of Non-Spherical Granular Materials. The University of Electro-Communications, Japan, http://ir.lib.uec.ac.jp/infolib/user_contents/9000000625/900000625.pdf
- Cole J. B., Island M., Weiland R. H. (1967): AIRCRAFT WING VARIABLE CAMBER LEADING EDGE FLAP. United States Patent Office, <https://docs.google.com/viewer?url=patentimages.storage.googleapis.com/pdfs/US3504870.pdf>
- Elmqvist H., Baldwin A.D., Dahlberg S. (2015): 3D Schematics of Modelica Models and Gamification. Proceedings 11th International Modelica Conference, Versailles, September 21-23, 2015.
- Elmqvist H., Olsson H., Goteman A., Roxling V., Zimmer D., Pollok A. (2015) Automatic GPU Code Generation of Modelica Functions. Proceedings 11th International Modelica Conference, Versailles, September 21-23, 2015.
- Ghandriz T. (2014): An algorithm for structural topology optimization of multibody systems, Master's thesis, Lund University. <https://sharepoint.srv.lu.se/sites/mimer/kursplanering/gu/S/PBstipendium/Nomineringar%202015/MATEMATIK%20Tohee%20Ghandriz.pdf>
- Ghandriz T., Führer C., Elmqvist H. (2015): Structural Topology Optimization of Multibody Systems. ECCOMAS Thematic Conference on Multibody Dynamics, Barcelona, Catalonia, Spain.
- Goteman A., Roxling V. (2015): GPU Usage for Parallel Functions and Contacts in Modelica, Master's thesis, Lund Institute of Technology, Lund, Sweden. (To be published)
- Gottland N. (2012): Make Geneva wheels of any size, <http://newgottland.com/2012/01/08/make-geneva-wheels-of-any-size/>
- Hippman G. (2003): An Algorithm for Compliant Contact Between Complexly Shaped Surfaces in MultiBody Dynamics. MultiBody Dynamics 2003, Lisbon, Portugal, http://www.pcm.hippmann.org/doc/eccomas03_hippmann.pdf
- Hippman G. (2013): Polygonal Contact Model. <http://www.pcm.hippmann.org/>

- Hofmann A., Mikelsons L., Gubsch I., Schubert C. (2014): Simulating Collisions within the Modelica MultiBody Library. Proceedings 10th International Modelica Conference, Lund, March 10-12, 2014, <http://www.ep.liu.se/ecp/096/099/ecp14096099.pdf>
- Karras T. (2012): Thinking Parallel, Part III: Tree Construction on the GPU. <http://devblogs.nvidia.com/parallelforall/thinking-parallel-part-iii-tree-construction-gpu/>
- Lavrov D. (2014), Collision Detection Using Z Order Curve Aka Morton Order. http://dmytry.com/texts/collision_detection_using_z_order_curve_aka_Morton_order.html
- Mueller R.K. (2015): OpenJSCAD.org User & Programming Guide. <http://openjscad.org/>
- Nassauer B., Kuna M. (2013): Contact forces of polyhedral particles in discrete element method. DOI 10.1007/s10035-013-0417-9, Springer Verlag.
- Nürnberg R. (2013): Calculating the volume and centroid of a polyhedron in 3d. <http://wwwf.imperial.ac.uk/~rn/centroid.pdf>
- Oestersötebier F., Wang P., Trächtler A. (2014): A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces. Proceedings 10th International Modelica Conference, Lund, March 10-12, 2014, http://www.ep.liu.se/ecp_article/index.en.aspx?issue=96;article=97
- Otter M., Elmqvist H., Diaz Lopez J. (2005): Collision Handling for the Modelica MultiBody Library. Proceedings 4th International Modelica Conference, Hamburg, March 7-8, 2005, pp. 45-53, <http://elib.dlr.de/12299/1/otter2005-modelica-collision.pdf>
- Segura C., Stine T., Yang J. (2013): Constructive Solid Geometry Using BSP Tree. https://www.andrew.cmu.edu/user/jackiey/resources/CSG/CSG_report.pdf
- Shabana. A. A. (2013): Dynamics of multibody systems. Cambridge university press.
- Shewchuk J. R. (2012): Lecture Notes on Delaunay Mesh Generation, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley.
- Simeon B. (2013): Computational flexible multibody dynamics: a differential-algebraic approach. Springer Science & Business Media.
- Tonon F. (2014): Explicit Exact Formulas for the 3-D Tetrahedron Inertia Tensor in Terms of its Vertex Coordinates. <http://docsdrive.com/pdfs/sciencepublications/jmssp/2005/8-11.pdf>
- Wallace E. (2012) csg.js. <http://evanw.github.io/csg.js/>
- Zimmer D. (2012): A Planar Mechanical Library for Teaching Modelica. Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany

Different Models of a Scaled Experimental Running Gear for the DLR RailwayDynamics Library

Christoph Schwarz Andreas Heckmann Alexander Keck

Institute of System Dynamics and Control, German Aerospace Center (DLR), 82234 Wessling
{Christoph.Schwarz, Andreas.Heckmann, Alexander.Keck}@dlr.de

Abstract

The DLR internal project “Next Generation Train” (NGT) deals with a high-speed train in a double-deck configuration. To realize the two continuous floors, a single wheel running gear configuration is selected. Equipped with independently rotating wheels instead of a usual wheel-set, a track guidance control becomes necessary. In terms of an advanced control and observer development the implementation of validated simulation models is absolutely essential. Therefore, the paper gives a short overview of the hardware of the scaled Experimental Running Gear on the DLR roller rig representing the NGT single wheel running gear. Using the DLR RailwayDynamics Library three different models of the running gear are implemented, which vary in complexity and can be used for different analysis methods. Finally, some significant simulation results of the particular simulation models are presented and discussed.

Keywords: railway vehicle dynamics, running gear, analytical modeling

1 Introduction

In 1985 a roller rig was established at DLR. Since then various aspects of the railway vehicle dynamics have been investigated using constantly advanced running gears. The current configuration represents the single axle running gear of the DLR internal project “Next Generation Train” (NGT) (Winter et al., 2011). This project is targeted on a high-speed train in lightweight design with high demands for energy efficiency and the passenger capacity. To achieve these goals the train is designed in a double-deck configuration with continuous floors on both levels. Therefore, the running gear is equipped with independently rotating wheels (IRW) mounted on an axle bridge and individually driven. To stabilize the running gear dynamics, which are unstable at higher speeds (Wickens, 2003), a mechatronic track guidance is applied to ensure the safe service of the train and to reduce the wheel and rail wear.

Regarding the synthesis of a model based control and

an extensive analysis of the running gear system, the development of an appropriate and validated but at the same time simple simulation model is essential. In a first step, the hardware of the scaled running gear on the DLR roller rig is described in Section 2. Furthermore, the paper establishes three different simulation models in Section 3 varying in the level of complexity and implemented using the DLR RailwayDynamics Library. Some simulation results as well as a comparison thereof are delineated in Section 4. Finally, in Section 5 an outlook to the future work is given together with conclusions.

2 Hardware of the running gear on the roller rig

The actual running gear operating on the roller rig is a scaled 1 : 5 version of the single axle running gear conceived for the intermediate wagons of the NGT. The major components of the Experimental Running Gear are the central frame, the two axle bridges, and the four wheels. Figure 1 illustrates the particular modules as well as the mechanical degrees of freedom relevant for the mechatronic track guidance control. Using the indices $i = f, r$ (front, rear axle bridge) and $j = r, l$ (right, left wheel), respectively, the DOFs are the lateral displacements y_f and y_r of the center point of the axle bridge P with respect to the railroad centerline, the yaw

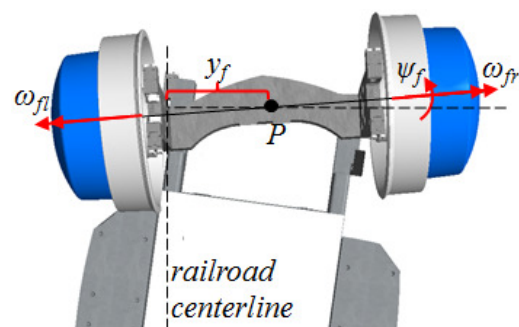


Figure 1. Half model of the running gear with mechanical degrees of freedom.

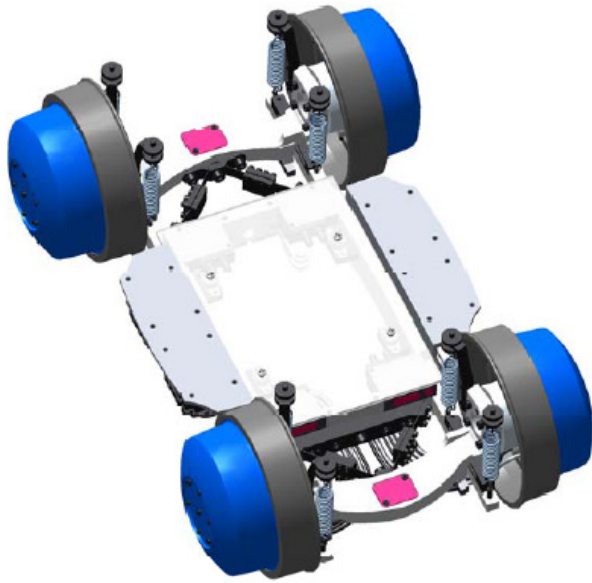


Figure 2. Detailed CAD model of the current running gear design.

angle of each axle bridge ψ_f and ψ_r and the four angular wheel velocities ω_{fr} , ω_{fl} , ω_{rr} and ω_{rl} .

The central frame is a screwed construction and very small manufacturing tolerances are demanded on the particular parts. In addition, the frame is mounted to the roller rig by a lemniscate guidance, that blocks the longitudinal motion of the running gear but allows for lateral and vertical motions as well as yawing. Laser sensors used for the measurements of y_i and ψ_i are attached to the frame just as the converters of the motors. Furthermore, each axle bridge is interconnected to the frame by two leaf springs as it can be seen in Figure 2. The springs act on the one hand as an axle bridge guidance with respect to the frame and on the other hand as vertical suspension. The adjustment of these springs in a V-shape guarantees a guidance free of clearance and allows radial steering of the axle bridges, what is necessary for the mechatronic track guidance. This leaf spring guidance concept and an improved cable guidance from the laser sensors and converters to the target PC are the outcome of the latest redesign of the running gear in 2014. Another enhancement of the newly constructed running gear are the low torsional stiffness of the frame and the additional vertical springs, that can be added in order to adapt for an optional, additional load. The vertical springs and the leaf springs lead to a rotational yaw stiffness between the axle bridges and the central frame, which has to be taken into account in the development of the track guidance control.

The IRWs are individually driven by permanent-magnet synchronous motors. In contrast to the real NGT setup, these in-wheel drives only have to deliver the control torque but not the traction torque, which is generated by the rollers. There are two independent control torques, one for the leading and one for the trailing axle bridge, since the torques of the right and the left motor

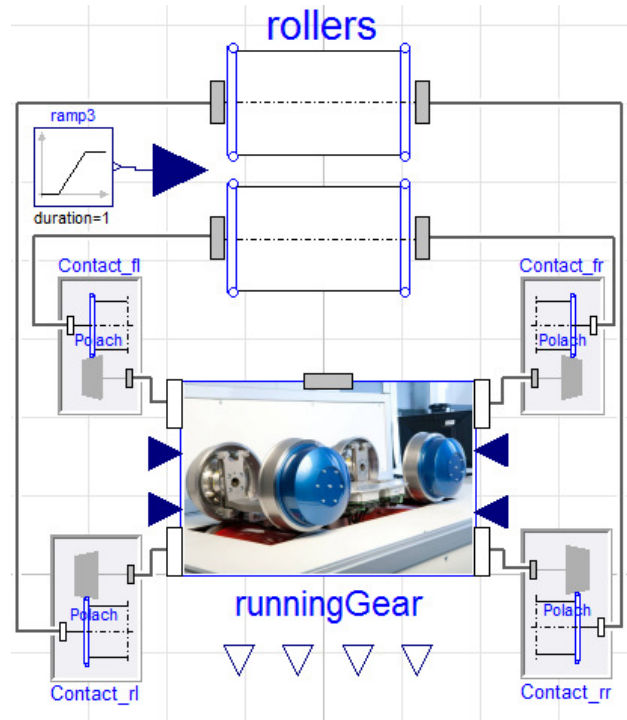


Figure 3. Modelica model of the running gear on the roller rig.

are equal in amount but opposite in direction. The two torques are calculated by separate, but identically parameterized controllers. The cascaded control structure implies an inner PD loop for the yaw angle and an outer PI loop for the lateral displacement.

3 Modeling aspects of the scaled running gear

Before describing the specific models with their characteristics some general aspects are pointed out that all three of them have in common. Firstly, the limitation of the actuator torque representing a non-linearity is modeled as part of the controller. Furthermore, the angular wheel velocities ω_{ij} are negative, since the angular roller velocity $\omega_R = \frac{v_R}{r_R}$ is positive in case of a positive v_R . Another common aspect of the models described hereafter is to idealize the wheel profile as perfectly conical with the cone angle d .

3.1 Detailed multibody Model

The multibody model of the running gear is divided into two parts: the running gear itself and the contact models of each wheel-rail pair, see Figure 3. According to (Heckmann et al., 2014) there are two options for the contact model: *Kalker's* linear theory and the theory formulated by *Polach*. Due to the more accurate contact formulation the latter theory is used to calculate the creep forces $\mathbf{f} = (f_x, f_y, l_z)^T$, with the longitudinal creep

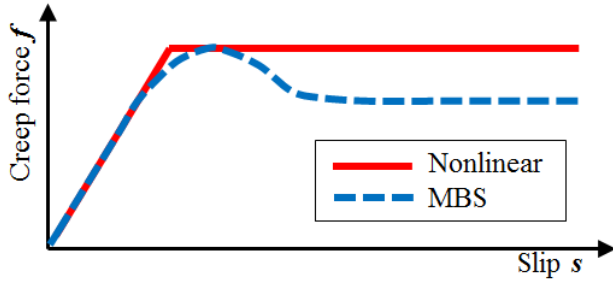


Figure 4. Creep forces in relation to the spin for the MBS and the nonlinear model (Knothe and Stichel, 2003).

force f_x , the lateral creep force f_y , and the torque l_z . The nonlinear relation between the creep forces and the slip $\mathbf{s} = (s_x, s_y, \phi_z)^T$, with the longitudinal slip s_x , the lateral slip s_y , and the spin ϕ_z , is illustrated in Figure 4. In addition, the shear modulus G and the *Kalker* coefficients C_{11} , C_{22} , C_{23} and C_{33} determine the calculation of \mathbf{f} . The *Kalker* coefficients depend on the semi-axes of the contact ellipse a and b and are stored in look-up tables in the contact modules.

The running gear model comprises all of the major components described in the previous section and is in turn divided into different substructures. On the top level the interfaces for the in- and outputs τ_{ij} , y_i and ψ_i are implemented, see Figure 3. Further parameters defined in this level are the wheel base e , the stiffness and the damping coefficient of the vertical spring/damper component as well as the body parameters of the central frame like its mass M and the moments of inertia. In a first sublevel the DOFs of the central frame are modeled using three rotational and two translational joints that are connected in series. The leading and the trailing axle bridges are represented by identical substructures. Considering the axle bridge structure the rotational stiffness k between the central frame and the axle bridge is modeled independently of the vertical spring. This component is positioned at the yaw joint of the axle bridge followed by the roll joint and the translational joint for the vertical displacement. Furthermore, the wheel gauge f , the nominal rolling radius of the wheels r_0 and the body parameters of the wheels and the axle bridge are defined. Another feature implemented in the axle bridge model is the transfer behavior from the requested controller torque to the actual angular wheel acceleration.

3.2 Nonlinear analytical Model

Since the overall aim is to generate a model that can be used for the development of a feed-forward as well as a feed-back control, another more simple model is necessary than the MBS model. Therefore, the complexity and the computational effort of the nonlinear model is reduced by carrying out some simplifications. First of all, the three rotational degrees of freedom of the central frame are locked, since for the true scale NGT rail-

way car with $e = 14\text{ m}$ and $f = 1435\text{ mm}$ the influences of these rotations might be diminutive anyway. Though, for a later application to the true scale NGT the influences of the mass and the inertias of the car body have to be investigated. Another simplified aspect is the neglect of the vertical stiffness between the axle bridge and the frame, so the vertical displacement of the central frame is $z_F = \frac{z_f + z_r}{2}$. In addition, the width of the continuous roller rails is considered to be infinitesimal regarding the calculation of the position of the contact patch on the wheel surface. Furthermore, *Kalker's* linear theory (Kalker, 1990) is used to calculate the creep forces \mathbf{f} in the wheel-rail contact. However, the maximum creep force is limited to $\mathbf{f}_{max} = F_N \mu$, with the normal wheel force F_N and the friction coefficient μ between wheel and rail, see Figure 4. To further decrease the computational effort the creep torque is neglected (Polach, 2000) and the *Kalker* coefficients are kept constant. Thus, the contact formulation is stated as (Knothe and Stichel, 2003)

$$\mathbf{f} = \begin{pmatrix} f_x \\ f_y \end{pmatrix} = \mathbf{K} \begin{pmatrix} s_x \\ s_y \\ \phi_z \end{pmatrix}, \quad \text{with} \quad (1)$$

$$\mathbf{K} = -abG \begin{pmatrix} C_{11} & 0 & 0 \\ 0 & C_{22} & \sqrt{ab}C_{23} \end{pmatrix}.$$

The mathematical description of the nonlinear dynamics is carried out using three coordinate systems: the inertial (index I), the body fixed (index ψ , located in the middle of the axle bridge) and the contact point coordinate system (index r, l) (Jaschinski, 1990). However, the transformation matrices \mathbf{A} converting these coordinate systems disregard the rotation of the wheels around the y -axis (Bremer, 1988).

The nonlinear model is deduced from the Euler-Lagrange equations

$$\frac{d}{dt} \left(\frac{\partial T}{\partial \dot{\mathbf{q}}} \right)^T - \left(\frac{\partial T}{\partial \mathbf{q}} \right)^T + \left(\frac{\partial V}{\partial \mathbf{q}} \right)^T = \mathbf{Q}, \quad (2)$$

with the generalized forces \mathbf{Q} , the time derivative $\dot{\mathbf{q}} = (\dot{y}_f, \dot{y}_r, \dot{\psi}_f, \dot{\psi}_r, \omega_{fr}, \omega_{fl}, \omega_{rr}, \omega_{rl})^T$ of the generalized coordinates \mathbf{q} and the kinetic and potential energy T and V , respectively. With the masses of the frame M and of an axle bridge including two wheels m , the moments of inertia of an axle bridge with respect to yawing B and of a wheel with respect to rolling C the kinetic energy T results in

$$T = \frac{M}{8} \left((\sum \dot{y}_i)^2 + (\sum \dot{z}_i)^2 \right) + \frac{m}{2} (\sum \dot{y}_i^2 + \sum \dot{z}_i^2) + \frac{B}{2} (\sum \dot{\psi}_i^2 + \sum \dot{\alpha}_i^2) + \frac{C}{2} (\sum \omega_{ij}^2). \quad (3)$$

In addition, the potential energy V is determined by two effects. The first is the potential V_s of the rotational spring with the stiffness k at the connection from the

frame to the wheel carrier and the second is the elevation energy V_e

$$V = V_s + V_e = \frac{k}{2} \psi_f^2 + \frac{k}{2} \psi_r^2 + (2m + M) g z_F. \quad (4)$$

In a next step, the velocities and displacements, needed for the calculation of the kinetic and potential energy, are determined. Since y_i , ψ_i and ω_{ij} are generalized coordinates or time derivatives thereof, only α_i and z_i have to be substituted for \mathbf{q} and $\dot{\mathbf{q}}$, respectively. The kinematic relation between the rotation of the axle bridges about the x -axis and their translation along the y -axis is stated as (Jaschinski, 1990)

$$\alpha_i \approx \tan \alpha_i = -\frac{d}{\frac{f}{2} - r_0 d} y_i = -\Gamma y_i. \quad (5)$$

The vertical movements z_i are on the one hand characterized through the yaw motions of the axle bridges and on the other hand through their lateral displacements. These influences can be treated separately (Jaschinski, 1990), so that the vertical displacements of the axle bridges are

$$z_i = z_i(\psi_i) + z_i(\alpha_i) = \frac{df}{2} \left(\frac{1}{\cos \psi_i} - 1 \right) + y_i \tan \alpha_i. \quad (6)$$

The calculation of the generalized forces will be described through the right wheel of the leading axle bridge but can easily be transferred to the other wheels. The correlation between \mathbf{Q}_{fr} , the absolute and angular velocities of the wheel at the contact point and the creep force \mathbf{f}_{fr} is

$$\mathbf{Q}_{fr} = \left(\frac{\partial \mathbf{v}_{fr}}{\partial \dot{\mathbf{q}}} \right)^T \mathbf{f}_{fr} + \left(\frac{\partial \boldsymbol{\Omega}_{fr}}{\partial \dot{\mathbf{q}}} \right)^T \begin{pmatrix} 0 \\ \tau_f \\ 0 \end{pmatrix}. \quad (7)$$

Thus, the required velocities are deduced as (Jaschinski, 1990)

$$\mathbf{v}_{fr} = \mathbf{A}_{\psi I} \begin{pmatrix} 0 \\ \dot{y}_f \\ \dot{z}_f \end{pmatrix} + \left(\boldsymbol{\psi} \boldsymbol{\Omega}_{fr} \times \begin{pmatrix} x_{fr} \\ y_{fr} \\ r_{fr} \end{pmatrix} \right), \quad (8)$$

with $\boldsymbol{\psi} \boldsymbol{\Omega}_{fr} = \begin{pmatrix} 0 \\ \omega_{fr} \\ 0 \end{pmatrix} + \begin{pmatrix} \cos \psi_f \dot{\alpha}_f \\ -\sin \psi_f \dot{\alpha}_f \\ \dot{\psi}_f \end{pmatrix}$.

The actual rolling radius r_{fr} is determined through the distance y_{fr} from the axle bridge center to the contact point along the body fixed y -axis

$$r_{fr} = \bar{r} - d \cdot y_{fr}, \quad \text{with} \quad \bar{r} = r_0 + \frac{df}{2}. \quad (9)$$

Furthermore, y_{fr} as well as the contact point shift x_{fr} are dependent on the yaw angle and y_{fr} is additionally dependent on the lateral displacement of the particular axle bridge.

At last, the creep force is calculated using (1), with the contact ellipse semi-axes a and b and the force $F_{N,fr}$ normal to the contact patch (Popp and Schiehlen, 2010)

$$\sqrt{ab_{fr}} = \sqrt[3]{3 \frac{F_{N,fr}(1 - \kappa)E_g}{2\pi(A + D)G\sqrt{g_v}}}. \quad (10)$$

A and D are geometrical parameters determined by the curvature of the contacting bodies in the vicinity of the contact patch and therefore they vary with the yawing and the lateral displacement of the axle bridges. Nevertheless, A and D as well as the associated, dimensionless parameters E_g and g_v are set to a fixed value. Finally, the slip and spin of the nonlinear running gear model is

$$\begin{pmatrix} s_{x,fr} \\ s_{y,fr} \\ \phi_{z,fr} \end{pmatrix} = \frac{1}{v_R} \begin{pmatrix} r v_{x,fr} + v_R \cos \psi_f \\ r v_{y,fr} - v_R \sin \psi_f \\ r \Omega_{z,fr} \end{pmatrix}. \quad (11)$$

3.3 Linear analytical Model

On the basis of the model described in the previous section a linear analytical model in the form

$$\mathbf{T} \dot{\mathbf{x}} = \mathbf{B} \mathbf{u} + (\mathbf{A}_1 + \mathbf{A}_2) \mathbf{x} \quad (12)$$

is generated, with $\mathbf{x} = (y_i, \psi_i, \dot{y}_i, \dot{\psi}_i, \Delta \omega_{ij})^T$ and $\mathbf{u} = (\tau_f, \tau_r)^T$. To receive the linear state space representation, $\Delta \omega_{ij} = \omega_{ij} - \omega_0$ is substituted for ω_{ij} (Goodall and Hong, 2000), with $\omega_0 = -\frac{v_R}{r_0}$. Though the system gives no direct information about the angular wheel velocities, the accurately measurable velocity v_R allows to recalculate ω_{ij} .

The linearized equations of motion are presented in equation (13) and are discussed in the following. Due to the symmetry of the running gear the dynamics of the leading and the trailing axle bridge are determined in the same way. First of all, the differential equations of the lateral displacement show a coupling of the two axle bridges. In the scaled environment this mutual influence is quite small, since the mass of the axle bridge and the connected wheels m is more than two times larger than the mass of the frame M . Regarding a 1:1 railway vehicle the mass ratio is vice versa and the coupling between front and rear axle bridge is more distinctive.

The part of the generalized forces referring to the torque τ_i constitutes the input matrix \mathbf{B} . The next term is the linearization of the derivative of the potential energy and represents the matrix \mathbf{A}_1 . This vector shows clearly the two components of V , namely the elevation energy and the spring potential. In addition, the two influences on the elevation energy generated by the yaw and lateral motion, respectively, can be seen.

In terms of the linearization, the limitation of the creep forces at higher slip values illustrated in Figure 4 is repealed in this model. Nevertheless, the gap between the nonlinear and the linear creep force calculation can be

$$\begin{pmatrix} \left(\frac{M}{4} + m + B\Gamma^2 \right) \ddot{y}_f + \frac{M}{4} \ddot{y}_r \\ \frac{M}{4} \ddot{y}_f + \left(\frac{M}{4} + m + B\Gamma^2 \right) \ddot{y}_r \\ B\ddot{\psi}_f \\ B\ddot{\psi}_r \\ C\Delta\dot{\omega}_{fr} \\ C\Delta\dot{\omega}_{fl} \\ C\Delta\dot{\omega}_{rr} \\ C\Delta\dot{\omega}_{rl} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \tau_f \\ -\tau_f \\ \tau_r \\ -\tau_r \end{pmatrix} + \begin{pmatrix} (2m+M)g\Gamma y_f \\ (2m+M)g\Gamma y_r \\ - \left[(2m+M)\frac{gd_f}{4} + k \right] \psi_f \\ - \left[(2m+M)\frac{gd_f}{4} + k \right] \psi_r \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 2\bar{c}_{22}\eta \left(\psi_f - \frac{\zeta}{v_R} \dot{y}_f \right) - \frac{4\bar{c}_{23}d}{fr_0} \left(\eta + \Gamma \frac{fd}{2} \right) y_f + \frac{\bar{c}_{23}\eta}{v_R} [2\dot{\psi}_f + d(\Delta\omega_{fr} - \Delta\omega_{fl})] \\ 2\bar{c}_{22}\eta \left(\psi_r - \frac{\zeta}{v_R} \dot{y}_r \right) - \frac{4\bar{c}_{23}d}{fr_0} \left(\eta + \Gamma \frac{fd}{2} \right) y_r + \frac{\bar{c}_{23}\eta}{v_R} [2\dot{\psi}_r + d(\Delta\omega_{rr} - \Delta\omega_{rl})] \\ -\bar{c}_{11} \left[\frac{fd\zeta}{r_0} y_f + \frac{fr_0}{2v_R} \left(\frac{f}{r_0} \dot{\psi}_f - \Delta\omega_{fr} + \Delta\omega_{fl} \right) \right] - \bar{c}_{23}fd \left(\frac{1}{r_0+r_R} - \frac{d^2}{r_0} \right) \psi_f \\ -\bar{c}_{11} \left[\frac{fd\zeta}{r_0} y_r + \frac{fr_0}{2v_R} \left(\frac{f}{r_0} \dot{\psi}_r - \Delta\omega_{rr} + \Delta\omega_{rl} \right) \right] - \bar{c}_{23}fd \left(\frac{1}{r_0+r_R} - \frac{d^2}{r_0} \right) \psi_r \\ -\bar{c}_{11}r_0 \left[-\frac{d\zeta}{r_0} y_f - \frac{f}{2v_R} \dot{\psi}_f + \frac{r_0}{v_R} \Delta\omega_{fr} \right] - \frac{\bar{c}_{23}fd^2}{2(r_0+r_R)} \psi_f \\ -\bar{c}_{11}r_0 \left[\frac{d\zeta}{r_0} y_f + \frac{f}{2v_R} \dot{\psi}_f + \frac{r_0}{v_R} \Delta\omega_{fl} \right] + \frac{\bar{c}_{23}fd^2}{2(r_0+r_R)} \psi_f \\ -\bar{c}_{11}r_0 \left[-\frac{d\zeta}{r_0} y_r - \frac{f}{2v_R} \dot{\psi}_r + \frac{r_0}{v_R} \Delta\omega_{rr} \right] - \frac{\bar{c}_{23}fd^2}{2(r_0+r_R)} \psi_r \\ -\bar{c}_{11}r_0 \left[\frac{d\zeta}{r_0} y_r + \frac{f}{2v_R} \dot{\psi}_r + \frac{r_0}{v_R} \Delta\omega_{rl} \right] + \frac{\bar{c}_{23}fd^2}{2(r_0+r_R)} \psi_r \end{pmatrix}. \quad (13)$$

kept within tolerable limits with the help of the actuator saturation. Hence, the generalized forces \mathbf{Q}_f caused by the creep forces determine the matrix \mathbf{A}_2 . The vector in the second line of equation (13) describes this part of \mathbf{A} , using the dimensionless, geometrical parameters $\zeta = 1 + \Gamma r_0$ and $\eta = 1 + \Gamma \bar{r}$ and the extended *Kalker* coefficients $\bar{c}_{11} = abGC_{11}$, $\bar{c}_{22} = abGC_{22}$ and $\bar{c}_{23} = (ab)^{\frac{3}{2}} GC_{23}$. The parameters \bar{c}_{11} and \bar{c}_{22} have the dimension of a force and \bar{c}_{23} the dimension of a torque. Considering the coefficients related to $\Delta\omega_{ij}$ it becomes obvious that the model is just linear in case of a constant roller velocity v_R . Furthermore, the linear model is only valid for this specific v_R , since the system behavior strongly depends on this parameter, what will be substantiated in the following section. Another aspect is that for a real configuration with longitudinal rails instead of rollers, the terms reciprocally proportional to r_R vanish, because $r_R \rightarrow \infty$. Substituting common values for the parameters, it turns out that the influence of the spin is quite small in relation to the slip.

4 Results

After describing the three different models, they are compared in this section by the illustration of some substantial simulation results. At first, some analysis results of the linear model are presented to get more insight in the running gear dynamics. Figure 5 shows the eigenvalues of the linear model for $v_R \in [0.1 \frac{m}{s}; 50 \frac{m}{s}]$ with steps of $0.1 \frac{m}{s}$. Considering the similarity laws stated in (Jaschinski, 1990) this velocity range corresponds to

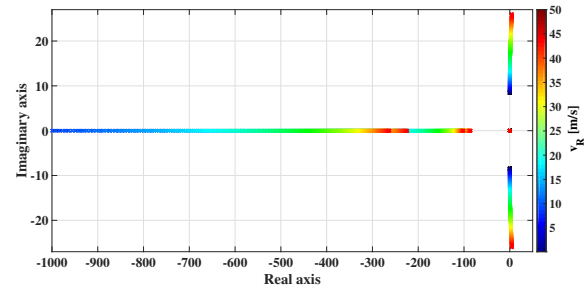


Figure 5. Eigenvalues of the linear model with varying roller speed v_R .

$[1 \frac{km}{h}; 400 \frac{km}{h}]$ in a 1:1 configuration. According to equation (13), there are four first order terms, namely the equations describing the angular wheel motions, and four second order terms, namely the lateral and the yaw motions. Due to the symmetry of the running gear the eigenvalues of the front and the rear axle bridge occur in pairs. One of this eigenvalue pairs is at low velocities approximately $-1.3 \cdot 10^5$, since the coefficients reciprocally proportional to v_R are very large in this case. In addition, it can be seen that with growing wheel velocity the real parts of all eigenvalues are moving in the positive direction, i. e. their dynamic behavior becomes slower. The detailed view of the area around the imaginary axis in Figure 6 shows that only two of the six eigenvalue pairs have got an imaginary part and consequently describe an oscillating behavior. This oscillating behavior called hunting motion is characteristic for railway vehicles with a conical or any nonlinear wheel profile.

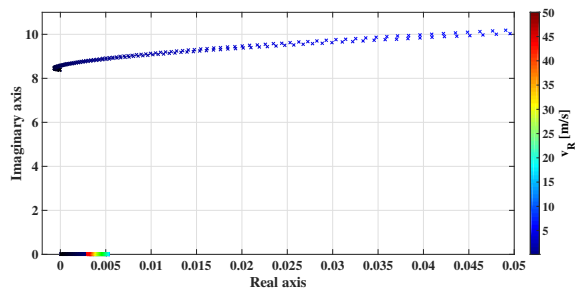


Figure 6. Unstable eigenvalues of the linear model with varying roller speed v_R .

Considering a conventional wheelset, the angular wheel velocities of the right and the left wheel are the same and in this way a self centering behavior occurs (Wickens, 2003). However, there is a critical velocity, that describes the maximum speed from which on the system is unstable. In (Dellmann and Abdelfattah, 2012) it was described that this hunting motion and the critical speed exist also for IRWs, what can be seen in Figure 6. In contrast to (Dellmann and Abdelfattah, 2012) the setup differs to some extent, e.g. the implementation of the yaw stiffness between the frame and the axle bridge. Because of this structural peculiarity, the eigenvalues describing the hunting motion possess already an imaginary part of about 8.4 at very low speeds. Another feature causing an instability is the potential energy, since a yaw motion of conical wheels comes along with a reduction of potential energy. This instability is characterized by the real eigenvalue in Figure 6.

In a next step, the transfer behavior from the input $u = \tau_f$ to the output $y = y_f$ of the real running gear is compared to the MBS model in Figure 7. The particular curves are created using a chirp signal as input (Saupe and Knoblach, 2012) and each is showing a nearly constant transfer behavior for frequencies of up to 1 Hz. The green curve represents the running gear configuration without the redesign enhancements described in section 2. It shows an analog trend in relation to the measured roller rig data but has got a constant offset. The same resemblance characterizes the model that takes the newly constructed leaf spring guidance into account and

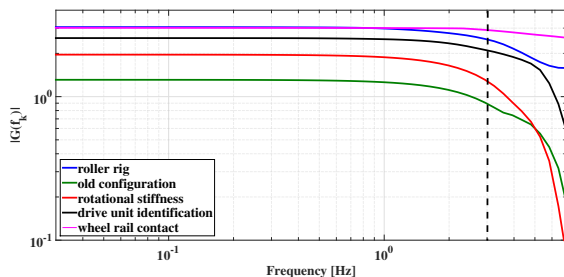


Figure 7. Measured and simulated transfer behavior of the running gear.

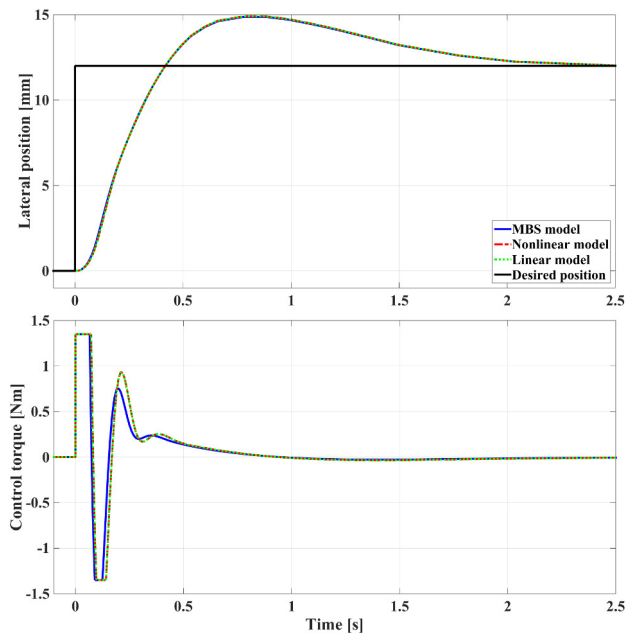


Figure 8. Desired and actual lateral displacement of the three simulation models at $v_R = 2 \frac{m}{s}$.

the model with identified drive units in addition. Finally, the MBS model that additionally has got adapted wheel/rail contact parameters, e. g. coefficient of friction, conforms the hardware running gear very well in the area up to 3 Hz. This adjustment shifts the drop, that can be seen for the other designs between 4 and 6 Hz, to higher frequencies. Nevertheless, another parameter-identification will be done to further match the dynamic behavior of the simulation model also in the frequency range above 3 Hz.

Since the detailed multibody model is validated with respect to the hardware running gear, the analytical models are compared to this model in the following two scenarios. Both simulations comprise a step of the desired lateral position so that the actuator torque reaches its limitation. Considering the applied roller speed the running gear with IRWs is an unstable system, so in consequence Figure 8 presents the results of the controlled running gear at a low velocity $v_R = 2 \frac{m}{s}$. The lateral displacements of the three models in Figure 8 show a very good match and the corresponding results of the control torque verify the conformity of the three models. This means that congruent positions without just as congruent torques would not approve the model conformity at all. The control torques of the linear and the nonlinear model are nearly identical, so the nonlinearities in the running gear dynamics might be insignificant at least for the selected level of complexity in section 3.2.

Furthermore, a test scenario illustrated in Figure 9 is recorded at a three times higher wheel velocity than in Figure 8. These simulation results show approximately the same distinguished conformity of the three models, though a slight deterioration at $t = 0.2$ s can be noticed

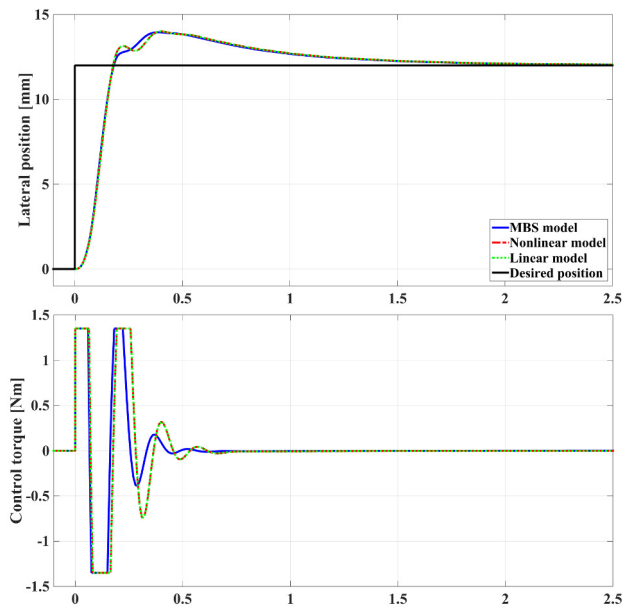


Figure 9. Desired and actual lateral displacement of the three simulation models at $v_R = 6 \frac{\text{m}}{\text{s}}$.

because of the higher speed. This might be the outcome of the neglected DOFs, what is tolerated for the sake of a less complex model. Nevertheless, the results of the linear and the nonlinear model are as identical as in the low speed simulation affirming the validity of the linearization also for higher wheel velocities.

5 Conclusions and Outlook

Based on the scaled 1:5 running gear on the DLR roller rig three simulation models have been established. First of all, a detailed multibody model has been described that has been used and validated in former works at DLR. Furthermore, a nonlinear analytical model with a reduced complexity has been deduced in detail. The third implemented and tested running gear model is a linearization of the nonlinear analytical model. The comparison of the results of the three models approves their validity and enables their use in the development of advanced control and observer concepts. In this context, the analytical models will be inverted and used in a feed-forward control concept (Heckmann et al., 2015). In addition, the excellent conformity of the linear and the nonlinear models facilitates the application of well-known linear analysis and control synthesis methods. Therefore, a plot of the eigenvalues has been illustrated as one of the results of the linear analysis of the running gear system and some characteristics of the dynamics of a railway vehicle with IRWs have been described.

Another aspect that can be scrutinized using the generated simulation models is a new sensor concept, since the laser sensors used for the measurement of y_i would not accurately work in a real application due to dirt. Regarding this, force and torques sensors are already installed at

the wheel mounting and shall after some further investigations replace the laser sensors. Finally, one part of the future work is to integrate the analytical models into the DLR RailwayDynamics Library to provide an environment for an advanced control development for railway systems.

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

- H. Bremer. *Dynamik und Regelung mechanischer Systeme*. Teubner-Verlag, Stuttgart, 1988.
- T. Dellmann and B. Abdelfattah. Comparison of dynamic properties of a conventional wheelset and an independently rotating wheelset - a theoretical contribution to an almost forgotten technology. *ZEVrail*, 136(10):380–390, 2012.
- R. Goodall and L. Hong. Solid axle and independently-rotating railway wheelsets - a control engineering assessment of stability. *Vehicle System Dynamics*, 33(1):57–67, 2000.
- A. Heckmann, A. Keck, I. Kaiser, and B. Kurzeck. The foundation of the DLR railwaydynamics library: the wheel-rail-contact. In *10th International Modelica Conference*, 2014.
- A. Heckmann, C. Schwarz, T. Bünte, A. Keck, and J. Brembeck. Control development for the scaled experimental railway running gear of DLR. *Vehicle System Dynamics*, 2015. to appear.
- A. Jaschinski. On the application of similarity laws to a scaled railway bogie model. Forschungsbericht DLR-FB 90-06, DLR, Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, Germany, 1990.
- J. Kalker. *Three-dimensional elastic bodies in rolling contact*, volume 2. Springer, 1990.
- K. Knothe and S. Stichel. *Schienenfahrzeugdynamik*. Springer, Berlin, 2003.
- O. Polach. A fast wheel-rail forces calculation computer code. *Vehicle System Dynamics*, 33:728–739, 2000.
- K. Popp and W. Schiehlen. *Ground Vehicle Dynamics*. Springer, 2010.
- F. Saupé and A. Knoblach. Design of excitation signals for the closed loop identification of industrial robots. In *IEEE International Conference on Control Applications*, 2012.
- A. H. Wickens. *Fundamentals of Rail Vehicle Dynamics: Guidance and Stability*. Swets & Zeitlinger, Lisse, NL, 2003.
- J. Winter, E. Mittelbach, and J. Schykowski, editors. *RTR Special - Next Generation Train*. Eurailpress, DVV Media Group, 2011.

Efficient Compilation of Large Scale Dynamical Systems

Federico Bergero^{1,2} Mariano Botta² Esteban Campostrini² Ernesto Kofman^{1,2}

¹CIFASIS, CONICET, Argentina {bergero, kofman}@cifasis-conicet.gov.ar

²FCEIA, UNR, Argentina {marianoabotta, lesteban22}@gmail.com

Abstract

In this work, we present a novel methodology to efficiently compile large scale dynamical systems described as Modelica models, and its implementation in a prototype Modelica Compiler called ModelicaCC. The methodology allows to perform the different stages of the compilation process without expanding the content of repetitive structures so the resources (CPU time and memory) used by the compiler result independent on the model size. Besides introducing the methodology with their algorithms and the implementation in the ModelicaCC compiler, we analyze their efficiency comparing its performance with that of OpenModelica in different large scale models.

Keywords: Modelica Compilers, Large Scale Models, Tarjan Algorithm, Model Flattening

1 Introduction

Modelica (Fritzson, 2004) is an object-oriented, equation-based language for representing continuous and hybrid models. Modelica provides a standardized way to model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power, or process-oriented subcomponents.

The simulation of these models requires some transformations. First, classes and connections amongst them are removed obtaining a *flat* model. A flat model yields a Differential Algebraic Equation (DAE) system which must be then sorted and converted into an Ordinary Differential Equation (ODE) system. From the ODE representation, C code is generated and compiled together with the ODE numerical integration method. This pipeline is performed by the Modelica compilers.

Frequently, Engineers and researchers of different domains need to simulate large scale models, which are usually the result of connecting together several identical components in repetitive structures. Examples of these models appear in Smart Grids, spatial discretization of Partial Differential Equations (PDE), etc. Although the Modelica language do allow to represent these large scale models in a convenient way, Modelica compilers

fail to complete the compilation pipeline when the size of the models grows beyond a few thousand of components. Thus, efficient handling of large scale models is an important topic in the modeling and simulation community (Cellier et al., 2013).

In this article we present novel algorithms to address the compilation of large scale Modelica models. The idea behind them is to exploit the repetitive nature of large scale models and perform all the mentioned transformations (flattening, sorting and code generation) without expanding the model arrays. Also, as outlined in (Stavaker, 2011), preserving the iterative equations until the code generation phase enables the use of parallel simulation techniques that would otherwise be useless.

We present also prototype implementations for these algorithms and compare their performance against OpenModelica compiler.

The work is organized as follows: Section 2 provides the main concepts used along the rest of the article. Then, Sections 3 and 4 introduce the novel algorithms developed for the flattening and causalization stages. After that, Section 5 presents the ModelicaCC compiler, describing its architecture and components. Finally, a comparative study of the Compiler performance on large scale systems is performed in Section 6 and the conclusions are presented in Section 7.

2 Background

In this section we first describe the process for simulation of Modelica models and we outline the problems faced when compiling large scale systems. Later we present the tool we use for generating C code and simulating, and finally we review some works related to this article.

2.1 Modelica Compilers

As mentioned earlier, Modelica models require different transformations before being simulated. First, a flattening stage is responsible for converting the Modelica model into an equivalent model without classes (inheritance and composition), converting also the `connect` equations into equalities. After this stage, the **flat** model only contains variables of basic types (real, integer,

boolean, etc) and equations (and algorithms) representing a hybrid DAE system.

This DAE system is then converted into an ODE system so it can be simulated by ODE solvers (Runge-Kutta, DASSL, DOPRI, etc). This conversion is divided in two sub-steps. First, if the problem has high index, an index reduction algorithm (such as Pantelides (Pantelides, 1988)) is applied. Then, the equations are horizontally and vertically sorted by a matching algorithm (such as Tarjan (Tarjan, 1972)) and the ODE system is obtained.

Then, an optimization stage is applied removing trivial equations (like $a = b$). Finally, C language code is generated and compiled together with the numerical solver obtaining an executable program that simulates the model.

Figure 1 shows a typical pipeline found on most Modelica tools, like OpenModelica (Fritzson et al., 2005), Dymola (Brück et al., 2002) and others.

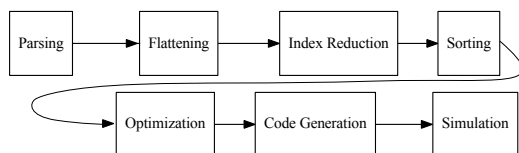


Figure 1. Pipeline of compilation

2.2 Modelica and Large Scale Models

With the availability of more powerful simulation platforms, models that in the past were dozen or hundreds of variable are increasing their size to thousands or millions of variables (Cellier et al., 2013). This imposes some new challenges on the modeling and simulation community. First we must have ways of modeling these huge systems. Modelica seems to be a good choice for that. By their nature, large scale models are rarely developed by extension, i.e. they are not handwritten. In general they possess some repetitive structure that makes them easier to be described by comprehension.

Modelica is well fitted for this kind of description. It allows the modeler to replicate components and connect them in a regular fashion. Thus a large scale model can be written with a short Modelica description. This is usually done using the `for` iterative equation for behavior description and array variables for states.

However, problems appear in the different stages of the compilation pipeline. When flattening large scale models, most Modelica tools perform what is known as loop unrolling, i.e. `for` equations are replaced by their equivalent scalar equations. In this stage array variables are also expanded into several scalar variables.

Consider for instance the lumped model of a LC Transmission Line depicted in Fig.2. This model can be described by the Modelica code of Listing 1.

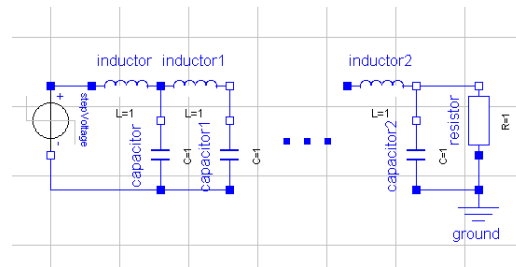


Figure 2. LC Line model

Listing 1. Hierarchical Modelica model of the LC Line example

```

model lcline
  import Analog = Modelica.Electrical.Analog ;
  model lcsection
    Analog.Interfaces.Pin pin2 ;
    Analog.Basic.Inductor i ;
    Analog.Interfaces.Pin pin ;
    Analog.Interfaces.Pin pin1 ;
    Analog.Basic.Capacitor c ;
  equation
    connect(c.p , pin1) ;
    connect(pin , i.p) ;
    connect(i.n , c.p) ;
    connect(c.n , pin2) ;
  end lcsection ;

```

```

    Analog.Basic.Resistor r ;
    Analog.Basic.Ground g ;
    constant Integer N = 100 ;
    lcsection[N] lc ;
    Analog.Sources.ConstantVoltage s ;
  equation
    connect(r.n , g.p) ;
    connect(s.n , g.p) ;
    connect(s.p , lc[1].pin) ;
    connect(r.p , lc[N].pin1) ;
    connect(g.p , lc[N].pin2) ;
    for i in 1:N - 1 loop
      connect(g.p , lc[i].pin2) ;
      connect(lc[i].pin1 , lc[i + 1].pin) ;
    end for ;
  end lcline ;

```

If we ask OpenModelica to flatten this model, the following code is obtained:

Listing 2. Flat model without arrays

```

class lcline
  constant Integer Size = 100 ;
  Real lc[1].c.v ; Real lc[1].c.i ;
  parameter Real lc[1].c.C ;
  ...
  Real lc[100].c.v ; Real lc[100].c.i ;
  parameter Real lc[100].c.C ;
  ...
  equation
    lc[1].c.i = lc[1].c.C * der(lc[1].c.v) ;
    ...
    lc[100].c.i = lc[100].c.C * der(lc[100].c.v) ;
    ...
  end lcline ;

```

In this code, we can see several equations like `lc[1].c.i=lc[1].c.C*der(lc[1].c.v)`, `lc[2].c.i=lc[2].c.C*der(lc[2].c.v)`, etc. coming from the `Capacitor` class inside each instance of the `lcsection` model.

That way, the cost of flattening this model is (at least) linear w.r.t. the `Size` parameter. Moreover, this expanded version will impose a huge burden on the successive stages of the compilation process as they must now deal with a large description.

In this work we argue that for efficient compilation of large scale models, all the steps involved in the process must be performed without expanding the model at all. Therefore in each step we deal with a large scale model but described in a short Modelica code.

2.3 μ -Modelica and QSS Stand-Alone Solver

The QSS Stand-Alone Solver (Fernández and Kofman, 2014) is an open source tool for continuous system simulation. The solver has a complete implementation of the QSS methods (Cellier and Kofman, 2006) and it includes also DOPRI and DASSL (Petzold, 1983) solvers.

The models must be described as a hybrid ODE system using a subset of the Modelica language called μ -Modelica. μ -Modelica contains the basic Modelica statements that allow to represent hybrid ODE systems as they are used by numerical ODE solvers, but using Modelica syntax instead of C or FORTRAN language. That way, the μ -Modelica code is easily translated by the QSS solver into C code and compiled together with the numerical integration method of choice.

A remarkable feature of this tool is that it does not unroll `for` loops in the generated C code. This has two up-sides, first the computational cost of the translation does not depend on the size of the loop (hence the size of the model), second this also holds for the compilation of the generated C code.

ModelicaCC compiler uses the QSS solver for the last stages of the compilation process (C code generation and simulation). Thus, the goal of the previous stages is to translate a Modelica model into μ -Modelica.

2.4 Related Work

Some work has been done regarding the compilation of large scale Modelica models. Studies have been developed testing how compilers work on large scale models recognizing their limitations on that area (Frenkel et al., 2011; Sezginer, 2014-2015; Casella, 2015). In (Jens Frenkel, 2012) particularly, the authors study different causalization (matching) algorithms applied to large scale models and conclude that the PF+ algorithm (by Duff) is the best choice as it achieves linear performance on the tested cases.

In (Zimmer, 2009) Zimmer also presents the problem compiling large scale models of current Modelica compilers. There the author proposes a solution based on the preservation of module structure during the compilation phase. This is achieved through partial flattening and causalization.

More closely related work was presented in (Arzt et al., 2014), where the authors explore the idea of finding repetitive structure on the incidence graph to efficiently apply Pantelides index reduction algorithm without dealing with flattening nor causalization. Therefore both works are complementary. The present work relates to this article in the sense that we also try to exploit the repetitive structures, but in our case, we develop new algorithms (in Section 3 and 4) to be applied on the an extended graph.

A first attempt to preserve array variables and iterative equations was presented in (Stavaker et al., 2010; Stavaker, 2011). There, the authors modified the flattening and causalization algorithm of OpenModelica to allow slice variables (such as `a[1:10]`) to be treated as a single variable. The modified algorithm does not correctly flattens hierarchical models and does not handle irregular definition of variables. As we will see, our proposal deals with more general and realistic cases (both while flattening and sorting).

3 Flattening Algorithm

This section presents an algorithm to efficiently flatten large scale models, without expanding equations and array variables. This stage is divided in two steps. The first one actually flattens the model and the second one removes the connect statements by the corresponding equations.

3.1 Class Flattening Stage

This stage of the flattening algorithm deals with class flattening leaving connections (and connectors) untouched. The algorithm follows the principles of most Modelica compilers, but with a special treatment for arrays of variables.

For example, when flattening the model shown in Listing 1 most Modelica tools will expand the array `lc.c.v` into `Size` variables and generate `Size` equations of the form `lc[1].c.i=lc[1].c.C*der(lc[1].c.v)`. While the result is correct, it will produce a large description for the successive stages of the compilation pipeline.

What we propose is to avoid expanding the arrays to generate the flat model. In the transmission line example of Listing 1, this can be done as follows:

- First flatten model `lcsection` by flattening their components `Capacitor` and `Inductor`. This

will result in a model with basic type variables.

- For flattening the array `lcsection lc[Size]` we proceed as follows:
 - For each variable in the flat model `lcsection` we define an array of size `Size`, prefixing the name of that variable with `lc_`.
 - We change the modifications (if any) on the new array variable.
 - We wrap each equation in a `for` loop (with range `i in [1:Size]`) adding the `[i]` index expression on each use of the above defined variables. The same is done for algorithmic sections.
- Now we are ready to remove the `lc` instance of `lcline`.

The result of applying these rules is shown in Listing 3.

Listing 3. Flat model without expanding arrays

```

model lcline
  constant Integer Size=100;
  parameter Real lc_c_C[Size];
  Real lc_c_v[Size];
  Real lc_c_i[Size];
  ...
equation
  ...
  for i in 1:Size loop
    lc_c_i[i] = lc_c_C[i]*der(lc_c_v[i]);
  end for;
end lcline;

```

We see that the length of this flat version is independent of the `Size` parameter.

The idea sketched in this example can be applied to most arrays of classes. However, if the array has a non-regular definition (due to the usage of a *type redeclare* modification in some components of the array, for instance), then a special procedure should be followed. Anyway, those cases are beyond the scope of this work since most practical large scale models do show regularity in their class definitions.

3.2 Connection Replacement Stage

After the class flattening stage, we still have to remove connectors and convert the `connect` operators into equations. This process cannot be done locally (at the component level) since connections are inter-component operations. Thus, we have no choice but to solve this problem looking at the complete model.

A `connect` equation binds the variables of two connector instances. Modelica distinguishes between two kind of variables inside connectors:

Potential variables which are equalized for connected connectors.

Flow variables which are zero-summed for connected connectors and zero-equalized for unconnected ones.

When replacing a `connect` operator, the potential variables are easily handled (they are converted to an equation like `a=b`). However, flow variables require a special treatment since more than two connectors can be connected together resulting in a zero-sum of multiple terms. To generate these equations we must compute the set of all connectors that share a connection.

We can associate this issue to a graph theory problem as follows. First we build an undirected bipartite graph with one node for each `connect` operator and one node for each connector instance. Then, for each `connect` node, we add two edges linking it with the corresponding connector nodes. Then, computing the connection sets is analogous to finding the connected components of the graph. Several search algorithms, like Depth First Search (DFS) (Hopcroft and Tarjan, 1973), achieve this goal with linear complexity.

Connections on Large Scale Models After the class flattening stage, arrays are preserved appearing inside iterative equations arising from scalar equations wrapped in `for` loops. As mentioned above, the `connect` equations and connector variables are still part of the model and they should be replaced by their equivalent equations.

If these connections and connectors belong to replicated models, our purpose is to replace them by the corresponding equations preserving the arrays and the `for` loops. To this end, we propose to extend the graph theory procedure explained above representing arrays of connectors and connections by single nodes (with some additional information) in order to find the connected components on this *vectorized* graph.

3.2.1 Vectorized Connection Graph

We build the vectorized connection graph as follows

- For each `connect` operator we create an e node. Each operator inside a `for` loop counts as a single node.
- For each connector, we add a c node. An array of connectors also counts as a single node.
- We add two edges for each `connect` operator linking it with the two connectors involved. Each edge will have the range $P \subset \mathbb{N}$ of use of each connector (in the case it corresponds to an array) as a property.

3.2.2 Vectorized Connection Algorithm

Now we must find the connected components in the vectorized graph. A modified DFS algorithm is proposed for that goal. The idea is to gather multiple connected components while traversing the graph using the information available on the edge properties.

Before presenting the algorithm, we provide the following definitions:

Definition 1 Given a vectorized graph \mathcal{G} and a range $P_0 \subseteq \mathbb{N}$, we define $\mathcal{G}(P_0)$ as the subgraph resulting containing all the nodes of \mathcal{G} and only the edges with range P such that $P_0 \subseteq P$.

Definition 2 Given a vectorized graph \mathcal{G} , we say that a subset \mathcal{C} of its nodes is connected in the range P_0 if it is connected in the classic sense on graph $\mathcal{G}(P_0)$.

Definition 3 Given a vectorized graph \mathcal{G} , a node c , and a range P_0 , we say that the set of nodes \mathcal{C} is a connected component including c in the range P_0 if it is a connected component including c in the classic sense on graph $\mathcal{G}(P_0)$. We shall denote it $\mathcal{C} = CC(\mathcal{G}, c, P_0)$.

Definition 4 Given a range P_0 and a node c of a vectorized graph \mathcal{G} , we say that P_0 is a compact range of c if $CC(\mathcal{G}, c, P_0) = CC(\mathcal{G}, c, P)$ for all $P \subseteq P_0$.

Notice that if P_0 is a compact range of c , then the connected component (cc) is the same for each subrange of P_0 . This means that all the variables involved in the cc have the same connection structure in the range P_0 . Thus, they can be treated in the same way.

Thus, given a vectorized graph, finding a large compact ranges allows to gather multiple components on a single step. The following algorithm performs this step.

Given a node c , we initially take P_0 as the range of one of its edges, and then:

1. Remove all the edges whose range P has empty intersection with P_0 .
2. In the resulting graph, find the classic connected component \mathcal{C} ignoring the edge ranges.
3. If two nodes of the connected component \mathcal{C} contain an edge with range P such that $P \cap P_0 \neq P_0$ then P_0 is not a compact range. Thus, take $P'_0 = P \cap P_0$ and go back to step 1.
4. Otherwise P_0 is a compact range including node c of the vectorized graph. Moreover, $\mathcal{C} = CC(\mathcal{G}, c, P_0)$ is the connected component including c in the range P_0 .

Finally, the previous algorithm can be iteratively used to find all the connected components on compact ranges on a vectorized graph \mathcal{G} as follows.

1. Take a node c with at least one edge with non empty range. If none is found, all the connected components have been already computed.
2. Compute a compact range P_0 and the corresponding connected component $\mathcal{C} = CC(\mathcal{G}, c, P_0)$ for node c using the previous algorithm.
3. Add \mathcal{C} and P_0 as a new connected component to the result.
4. Remove the range P_0 from all the edges in \mathcal{C} and go back to step 1.

This algorithm provides a set of connected components with the corresponding ranges. For instance, in the model of Listing 1 one of the connected components will be:

$$\{g.p, lc[1 : Size].pin2, r.n, s.n\} \quad (1)$$

The presented algorithm is not linear as the original DFS since it visits many times the same node. Anyway, the fact that we have only one node per array and one node for each iterative equation makes this algorithm significantly faster than its scalar counterpart in large scale models. Additionally, it allows to generate equations preserving the arrays and `for` loop equations.

For simplicity and space reasons, we only introduced the basic algorithm which does not cover all cases. Scalar variables inside loops have a special treatment, so that the edges have their ranges covering the whole set \mathbb{N} . Also, connections binding arrays with different ranges also need a special treatment which involves translating the ranges while traversing the vectorized graph.

Equation Generation Once we have computed the connected components we must generate their equations. Given a connected component we do:

- For each potential variable in the connector we add an equality equation binding their value. In the case of ranged connectors we include a `for` equation.
- For each flow variable we add a zero-sum equation. In the case with range connectors we include a `sum` term for all the elements involved.

For example, the first connected component of Listing 1 would yield the following equations:

Listing 4. Iterative equations for solved components

```
// Potential variables
for i in 1:Size
  g_p_v = lc_pin2_v[i];
end for;
g_p_v = r_n_v;
g_p_v = s_n_v;
// Flow variables
g_p_i + sum(lc_pin2_i) + r_n_i + s_n_i = 0;
```

4 Causalization Algorithm

This section presents a novel algorithm to convert a DAE system into an ODE system specially tailored for large scale models. This procedure can be also expressed as a graph theory problem. The idea is to capture the relation between equations and variables as an undirected bipartite graph and then applying Tarjan's algorithm to find the strongly connected components. We will review a simplified version of this algorithm presented in (Cellier and Kofman, 2006).

4.1 Classical Tarjan's Algorithm

We start from a flat Modelica model with N equations and N unknown variables ¹.

Graph Creation

- For each equation we add a vertex e to the graph.
- For each unknown variable we add a vertex v to the graph.
- We add an edge (e, v) if unknown v is used in equation e .

We will assume that unknown variables in Modelica models are of type `Real`. Parameters, constants and discrete variables are not considered unknown for the continuous part of the system. Also, assuming the absence of singularities, variables that appear inside a `der` operator are considered state variables which are known (with the unknown being their derivatives).

Causalization The causalization algorithm proceeds as follows:

1. Each e vertex of degree 1 (i.e., having only one outgoing edge) can be made causal since that equation has only one variable. Number the equation with the lowest available number starting from 1, follow the edge to its corresponding v node and remove all edges connected to v . Finally remove vertexes e and v .
2. Each v vertex of degree 1 can be made causal since that variable appears in only one equation. Then number the vertex with the highest available number starting from N , follow the edge to its corresponding e node and remove all edges connected to e . Finally remove vertexes e and v .
3. If we have numbered all equations we have finished. Else go to step 1.

¹If the number of equations is different than the number of unknowns the problem cannot be converted to an ODE system.

If a vertex (e or v) with degree 1 is not found, then an algebraic loop or a higher order singularity might exist, and a different procedure should be followed.

The space complexity of the algorithm is $\mathcal{O}(E + V)$ (with V number of vertex and E number of edges). The time complexity is also linear (if care is taken to find vertexes of degree 1) since each step removes one pair of vertexes.

When the algorithm finishes, the results is a sorted list (sorted by the number we assigned during the algorithm) of pairs (e, v) meaning that variable v must be solved using that equation e . Finally we solve each variable in each equation and we obtain a model in an ODE form.

Tarjan on Large Scale Models In order to apply Tarjan's algorithm to a large scale model, we should first unroll the `for` equations. This step alone takes a computational cost (time and space) proportional to the number of equations inside the `for` loop. Then, building the bipartite graph requires to create one e vertex for each unrolled equation and one v vertex for each element in the array variables (since `a[1]` and `a[2]` are different unknowns) as well as the edges of the graph. Finally, the cost of applying Tarjan's algorithm to this graph grows linearly with the number of vertexes.

In order to avoid this, we propose a modified Tarjan algorithm that is applied directly on the non-expanded model. This way, we not only avoid the cost of the expansion but this also results in applying Tarjan's algorithm to a much smaller graph and producing a much shorter ODE system description.

4.2 Vectorized Graph Representation

Here we start with a model with N_E (scalar or `for`) equations and N_V (scalar or array) variables. We will assume that every `for` loop has a single equation in its body. If this is not the case, it can be split into multiple `for` loops with only one equation in their body.

Then, we build an augmented bipartite graph where each edge has two properties: an equation range $p_e \subset \mathbb{N}$ and an index range $p_v \subset \mathbb{N}$ (two sets of integer numbers).

The graph is built as follows:

- We create an e vertex for each equation. If an equation is inside a `for` loop, only one node is created.
- We create a v vertex for each variable. Arrays are treated as a single variable.
- We add an edge (e, v) if unknown v is used in equation e with their p_e, p_v properties computed as follows:
 - $p_e = \{1\}$ if the equation is not inside a `for` loop.
 - Otherwise, p_e is the range of the `for` loop.

- $p_v = \{1\}$ if the variable is a scalar.
- Otherwise, p_v is the set of indexes of the array that are used at equation e .

For example the equation

```
for i in 2:10 loop
  der(a[i]) = b[i-1];
end for;
```

would have two edges, one connecting to array `der(a)` with properties $p_e = 2 : 10, p_v = 2 : 10$ and another edge connecting the equation with the array `b` with properties $p_e = 2 : 10, p_v = 1 : 9$.

If we compare this graph with the one resulting of expanding arrays and loops, we can see that the vectorized graph collapses all the variable vertexes of the same array into a single macro-vertex and all the equation vertexes of a same loop into a single macro-vertex. Also, edges have been merged so that a multi-edge with $p_e = [2 : 10]$ is equivalent to nine simple edges.

We propose next a Vectorized Tarjan's Algorithm to be applied to these augmented graphs.

4.3 Vectorized Tarjan's Algorithm

A vectorized version of the Tarjan's algorithm explained above can be sketched as follows:

1. Each e vertex of degree 1 can be made causal since those equations have only one variable each. Number the equation with the lowest available number starting from 1, follow the edge with properties p_e^1, p_v^1 to its corresponding v node and remove p_e^1 from the index range p_v of every outgoing edge connected to v . Remove edges with empty index range ($p_v = \{\}$), remove vertex e , and finally remove v if it has no more edges.
2. Each v vertex of degree 1 can be made causal since those variables are used in only one equation each. Number the equation with the highest available number starting from N , follow the edge with properties p_e^N, p_v^N to its corresponding e node and remove p_e^N from the equation range p_e of every outgoing edge connected to e . Remove edges with empty equation range ($p_e = \{\}$), remove vertex v , and finally remove e if it has no more edges.
3. If the graph is now empty, we have finished. Otherwise, go back to step 1.

In rule 2, N is the number of scalar variables we would have if we expand all arrays. The reason is that there are cases in which the algorithm above may actually expand some or even all the arrays.

Simple scalar cases Let us analyze first how this algorithm works for models without arrays and iterative equations. Here, the vectorized graph will have one e vertex for each equation and one v for each variable and all the edges will have $p_e = p_v = \{1\}$. When we make causal a vertex (e or v) we must follow the corresponding edge and compute the set difference of p_e^1 or p_v^1 with those of all outgoing edges. That difference will always be the empty set since all edges have the same p_e, p_v thus we will always remove all edges. Therefore, the Vectorized Tarjan's Algorithm fails back to the classical algorithm for simple scalar models.

Vectorized cases Let us see what happens on models with arrays and iterative equations on the two rules we have.

In rule 1, we make causal an e vertex using an edge with $p_v = \{i_1, i_2, \dots, i_m\}$. That multi-edge represents m simple edges meaning that equation e involves that set of indexes of the array v associated with the v node. When we causalize e , we are making causal variables $v[i1], v[i2], \dots$ in a **single step**. Once those indexes are causalized, they become known variables to the remaining equations and we remove them from the index ranges p_v of the remaining outgoing edges of v .

The same analysis can be performed regarding rule 2.

Non Covered Cases The Vectorized Tarjan's Algorithm can fail to find a vertex with degree 1. The reasons here are the same as in the classical Tarjan's algorithm, either the model is structurally singular or the model has an algebraic loop. In the present work we do not handle this type of problems with the Vectorized algorithm. If this case is found, appropriate algorithms can be applied (such as Pantelides) on the expanded model.

The algorithm presented can be extended to deal with certain structures that, without having algebraic loops, result in graphs where all nodes have degree larger than 1. For the sake of simplicity we have not address this in the present work, but the algorithm can be easily extended to cover the case where a node has degree 2 or higher but there are some indexes (in p_e or p_v) that are only used in one outgoing edge.

Anyway, the presented algorithm is still able to sort many practical models, as we shall illustrate in Section 6.

Complexity and Result Analysis The complexity analysis in this case is not as simple as in the classical Tarjan's. The space complexity is now $\mathcal{O}(N_E + N_V)$ since the graph has that number of nodes. Here we are assuming that the properties of the edges are stored not by extension but by comprehension (since they are continuous range of integers).

Each step causalizes at least one variable (or one element in the case of arrays), so in the worst case the

time complexity is the same as in the classical Tarjan’s algorithm. On models showing regular structure (as in large scale models), the algorithm will causalize more than one variable in each step reducing the time complexity. Going further, additional conditions can be imposed on the model to assure that algorithm achieves a $\mathcal{O}(1)$ time complexity w.r.t. the model size.

Equation Generation When the algorithm finishes, the result is a sorted list of pairs (e, v) where each pair has the two ranges p_e, p_v of the edge used to causalize it. From here, Modelica code can be generated with the sorted and solved equations. An equation with range larger than one will be placed inside `for` loops in the range p_e , so that variable $v[i]$ is computed there for i in the range p_v .

5 The Modelica C Compiler

We have developed a Modelica C Compiler (or ModelicaCC) to test the algorithms presented above. The architecture of the compiler follows the usual pipeline of Figure 1. In this case, we decided that the input/output of each stage are valid Modelica models (except for the very last stage which produce C code).

Each stage converts the Modelica model fed as input into a “simpler” equivalent one. As we will be using the QSS Stand-Alone Solver (Section 2.3) for the C code generation and simulation, the goal of the previous stages is to obtain a valid μ -Modelica model.

One design goal of ModelicaCC is to reuse as much available code as possible. So, several open source libraries were used as part of the implementation. The C++ STL and Boost library were used for representing the AST and the Boost-Spirit library was used for parsing. Also, the GiNaC library was used for symbolic manipulation of equations and the Newton iteration implementation of the GSL library is used for solving algebraic loops. Graph algorithms were implemented with the Boost Graph Library.

Below, we shall briefly describe the stages of the ModelicaCC tool outlining the novel features.

5.1 Flattening Stage

The flattening stage of ModelicaCC implements the algorithm presented in Section 3.1 and 3.2. The input to this stage is a total hierarchical Modelica model. The transformation of the first stage are mainly implemented as AST Visitors using the Boost library. The algorithm presented in Section 3.2 is also implemented using the Graph Library from Boost.

The command `./flatter` performs this stage, producing a flat Modelica valid code, which in large scale models preserves arrays and `for` loops.

5.2 Alias Elimination

Usually, Modelica models contain hundreds of trivial equations of the form $a = b$, most of them coming from `connect` operators. In order to simplify the tasks of the successive stages (causalization and code generation), these alias variables are removed together with their binding equations, replacing then the removed variable by their corresponding alias.

In the ModelicaCC implementation we also remove array alias. For instance, an equation like $a[i] = b[i]$ in the body of a `for` loop in the complete range of both arrays can be removed together with one of those variables. This process allows removing a large number of equations and variables in a single step.

The command `./antialias` performs this task, removing aliases from a flat Modelica model and producing a flat alias-free Modelica model.

5.3 Reduction to μ -Modelica Syntax

The QSS Stand-Alone Solver accepts models described in a subset of the Modelica language called μ -Modelica, which has a reduced and restricted syntax.

At this stage, the Modelica statements that are not part of μ -Modelica are replaced by semantically equivalent expressions and operators.

The result of this stage is still an unsorted DAE system containing only μ -Modelica constructs.

The command `./mmo` performs this conversion, taking a flat Modelica model and producing an equivalent flat Modelica model with μ -Modelica supported syntax.

5.4 Causalization Stage

The causalization stage of ModelicaCC has two flavors:

- Classical causalization, where `for` loops are unrolled and array are expanded. This implementation is able to tackle algebraic loops using an external C function that solves them using the GSL library.
- Vectorized causalization, implementing the algorithm presented in Section 4 that attempts to preserve the arrays and `for` loops.

Both implementations use the Boost library for the implementation of the graph theory algorithms and the GiNaC library for symbolic equation manipulation.

The vectorized algorithm is used by default. In case it fails (due to an algebraic loop not yet handled), the classical algorithm is used.

The result of this stage is a causalized μ -Modelica model that can be translated into C and simulated by the QSS Solver.

The command `./causalize` performs this task, taking a flat unsorted μ -Modelica model and producing a sorted μ -Modelica system.

5.5 Simulation Code Generation

The QSS Solver is in charge of the last stage of the compilation pipeline, generating the C code from the μ -Modelica model, compiling it together with the numerical solver.

As it was already mentioned, the QSS Solver does not expand arrays or `for` loops producing a compact piece of C code that can be quickly handled by the C compiler (it uses the `gcc`, GNU Compiler Collection).

The QSS Solver allows to simulate the models with different numerical algorithms, including DASSL, DOPRI and all the QSS family.

6 Examples and Results

We report here the usage of the algorithms and tools developed on two large scale systems of varying size. In both cases, we analyze the computational cost related to each stage of the compilation pipeline, comparing it with that of OpenModelica.

We also report simulation CPU time in order to assert the correctness and quality of the resulting C code.

Benchmark Platform We used OpenModelica 1.9.3+dev (r25437) and the QSS Solver in version rev.1299 on an Intel i7-3770 CPU @ 3.40GHz with a 64 bits Ubuntu Linux with 8Gb of RAM.

On both tools, we will use DASSL as numerical integration solver with a tolerance of $1e^{-6}$.

6.1 One Dimensional Heat Transfer

This model is a 1D Finite Difference discretization of a PDE heat transmission problem taken from (Sezginer, 2014-2015).

Table 1 shows the CPU time consumed by each stage of the pipeline by OpenModelica and ModelicaCC. There, the columns are **F**lattening, **S**orting, **C**ode Generation, **C**ode Compilation, **S**imulation.

Table 1. Timing (in sec.) of the compilation stages for different sizes of the Heat Transfer model

Size	OpenModelica			ModelicaCC			
	F+S+C	CO	SIM	F	S+C	CO	SIM
10	1.82	0.8	0.002	0.07	0.05	0.06	0.003
100	2.1	2.23	0.01	0.07	0.05	0.06	0.016
1K	7.9	8.18	0.5	0.07	0.05	0.06	0.8
4K	57.9	11.1	7.5	0.07	0.05	0.06	16.9
10K	316.9	26.7	—	0.07	0.05	0.06	—

We note that this model is flat already, so in the ModelicaCC case it could be directly fed to the causalization stage (since the whole pipeline is valid Modelica code) but we measure the trivial flattening stage anyway.

In the last row, with 10000 sections, DASSL fails to simulate.

6.2 A LC Transmission Line

The LC Line example is the one presented in Listing 1. Each section of the line is made with components from the MSL while the connections between these sections was done by writing the Modelica code.

Table 2 reports the CPU time of the different stages of the compilation.

Table 2. Timing (in sec.) of the compilation stages for different sizes of the LC Line model

Size	OpenModelica			ModelicaCC			
	F+S+C	CO	SIM	F	S+C	CO	SIM
10	0.1	1.0	0.006	0.03	0.04	0.2	0.005
100	1.9	1.4	0.14	0.03	0.04	0.2	0.094
1K	61.8	—	—	0.03	0.04	0.2	55
10K	—	—	—	0.03	0.04	0.2	—

Starting from 1000 components, the C compiler fails to compile the code generated by OpenModelica. For 10000 components, OpenModelica fails to generate the C code.

Again, for 10000 components DASSL fails to simulate the system.

6.3 Result Analysis

From the two examples studied above we see that the proposed algorithms and the ModelicaCC implementation have a constant complexity (both in space and in time) w.r.t. the model size. This not only allowed us to compile the models faster but we can handle larger models.

Notice that OpenModelica fails in the second case for 10000 components (which in fact correspond to more than 100000 variables, since the model contains several algebraic variable arrays). In that case, even for 1000 components, the C code produced is so large that it cannot be handled by the C compiler. All these problems disappear with the ModelicaCC approach.

In all cases, the simulation times are similar for both tools and the simulation results (not reported here) have a negligible difference. This tells us that DASSL is receiving equivalent set of equations. Moreover, when DASSL fails to simulate, DOPRI works fine with the QSS Solver.

In this benchmark we have only fully analyzed the open source OpenModelica tool. Anyway, we have also run the experiments on demo versions of Dymola and Wolfram System Modeler, obtaining a similar behavior when the model increases in size.

7 Conclusions and Future Research

In this article we presented novel algorithms for efficient compilation of large scale Modelica models. The idea behind them is to exploit the repetitive nature found on this kind of models and to process them without expanding iterative equations and array variables.

The two algorithms for Flattening and Causalization (Sec. 3 and 4) were implemented in a prototype C compiler called ModelicaCC. A design goal of ModelicaCC was to have a Modelica-valid pipeline throughout all the process and to exploit existing open source libraries.

The study performed on two large scale models shows that the algorithms and their implementation have a constant cost w.r.t. the model size while other Modelica tools have a supra-linear cost (both in flattening and code generation/compilation). This allows to compile arbitrary large models with no extra cost.

There are many directions for future work. First, the Vectorized Tarjan's algorithm must be extended to deal with algebraic loops. For higher index models, our idea is to extend Pantelides algorithm in the same way done for Tarjan's in order to reduce the index of non-expanded models.

As mentioned before, one possible source of large scale models is the spatial discretization of PDE's. The algorithms in this work are presented for unidimensional arrays and non nested loops. They must be extended to higher dimension and nested loops for their application in 2D and 3D discretizations.

Finally, so far we have made experiments with a few large scale models. All algorithms and tools presented here have to be tested on more complex cases.

The ModelicaCC compiler is an open source tool hosted at www.sourceforge.net/projects/modelicacc/.

The models used in this article can be downloaded from www.fceia.unr.edu.ar/~fbergero/modelical15/.

References

- Matthias Arzt, Volker Waurich, and Jörg Wensch. Towards Utilizing Repeating Structures for Constant Time Compilation of Large Modelica Models. In *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT '14*, pages 35–38, Berlin, Germany, 2014.
- Dag Brück, Hilding Elmquist, Sven Erik Mattsson, and Hans Olsson. Dymola for multi-engineering modeling and simulation. In *Proceedings of Modelica 2002*, 2002.
- Francesco Casella. Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives. In *11th International Modelica Conference*, 2015.
- F. Cellier, X. F. Floros, and E. Kofman. The Complexity Crisis: Using Modeling and Simulation for System Level Analysis and Design. In *Proc. SimulTech 2013, 3rd International Conference on Simulation and Modeling Methodologies, Technologies, and Applications*, Reykjavik, Island, 2013.
- F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer-Verlag, New York, 2006.
- Joaquín Fernández and Ernesto Kofman. A Stand-alone Quantized State System Solver for Continuous System Simulation. *Simulation*, 90(7):782–799, July 2014. ISSN 0037-5497. doi:10.1177/0037549714536255. URL <http://dx.doi.org/10.1177/0037549714536255>.
- Jens Frenkel, Christian Schubert, Gunter Kunze, Peter Fritzson, Martin Sjolund, and Adrian Pop. Towards a Benchmark Suite for Modelica Compilers: Large Models. In *8th Modelica Conference*, 2011.
- Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-Interscience, New York, 2004.
- Peter Fritzson, Peter Aronsson, Hakan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Development Environment. In *Proceedings of the 46th Conference on Simulation and Modeling (SIMS'05)*, pages 83–90, 2005.
- John Hopcroft and Robert Tarjan. Algorithm 447: Efficient Algorithms for Graph Manipulation. *Commun. ACM*, 16(6):372–378, June 1973. ISSN 0001-0782. doi:10.1145/362248.362272. URL <http://doi.acm.org/10.1145/362248.362272>.
- Peter Fritzson Jens Frenkel, Gunter Kunze. Survey of appropriate matching algorithms for large scale systems of differential algebraic equations. In *9th Modelica Conference*, 2012.
- Constantinos C. Pantelides. The Consistent Initialization of Differential-Algebraic Systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988.
- L. R. Petzold. A description of DASSL: a differential/algebraic system solver. In *Scientific computing (Montreal, Quebec, 1982)*, pages 65–68. IMACS, New Brunswick, NJ, 1983.
- Kaan Sezginer. A Test Suite of Large Scalable Models for Modelica Tool Evaluation. Master's thesis, POLITECNICO DI MILANO, 2014-2015.
- Kristian Stavaker. *Contributions to Parallel Simulation of Equation-Based Models on Graphics Processing Units*. PhD thesis, Linkopings Universitet, 2011.
- Kristian Stavaker, Daniel Rolls, Jing Guo, Peter Fritzson, and Sven bodo Scholz. Compilation of Modelica Array Computations into Single Assignment C for Efficient Execution on CUDA-enabled GPUs. In *3rd EOOLT*, 2010.
- Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.
- Dirk Zimmer. Module-Preserving Compilation of Modelica Models. In *7th Modelica Conference*, 2009.

Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives

Francesco Casella

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,

francesco.casella@polimi.it

Abstract

State-of-the-art Modelica tools are very effective at converting declarative models based on differential-algebraic equations into ordinary differential equations. However, when confronted with large-scale models of distributed systems with a high number of states (1000 or more) or with large algebraic systems of equations (1000 or more unknowns), they face a number of serious efficiency issues, that hamper their practical use for system design. The paper analyses these issues in detail, points out strategies for improvement, and also introduces a library of scalable test models that can be used to assess existing tools, as well as to help developing advanced solution methods for large-scale systems.

Keywords: Modelica Compilers, Large-Scale Models, Efficient Simulation

1 Introduction

After almost 20 years from the first release of the Modelica language definition 1.0 (The Modelica Association, 1997), the Modelica language is well-established for system-level modelling tasks in many domains of engineering, such as automotive, robotics, mechatronics, energy, aerospace, in particular when multi-domain modelling is required.

To the best of the author's knowledge, based on published literature and personal experience, the standard work flow of state-of-the art Modelica tools can be summarised by the following steps, which are described in detail by Cellier and Kofman (2006).

1. (*Flattening*) The Modelica code is parsed; classes are expanded and instantiated, and eventually brought into the so-called flat form, i.e., a set of scalar hybrid differential-algebraic equations together with a set of scalar variables and parameters.
2. (*Causalisation*) Structural analysis of the differential-algebraic equations (DAEs) is performed, in order to solve them efficiently for the state derivatives and algebraic variables. This

process includes equation ordering (BLT transformation), may require symbolic index reduction, and usually involves extensive symbolic processing, as well as the use of advanced techniques such as tearing or reshuffling for solving sub-systems of equations efficiently. In most cases, the use of numerical solvers for linear and non-linear systems of algebraic equations is required.

3. (*Time integration*) The code which results from the previous step is linked to some well-tested, general-purpose dense Ordinary Differential Equation (ODE) solver, including root-finding algorithms to handle state events in the case of hybrid models.

In principle, step 2 is not strictly necessary, as DAEs resulting from step 1 could be solved directly using numerical DAE solvers. In practice, this is not standard practice for two reasons: one is that object-oriented Modelica models very often end up having index greater than 1, that are challenging to solve numerically, the other is that the above-sketched process is usually more numerically robust and easier to initialize than the direct solution of the nonlinear DAEs.

As to step 3, most Modelica models end up being stiff, because the modular way of building the models very often generates some very fast dynamic phenomena that, albeit maybe not of interest for the modeller, cannot be easily removed from the model, because they stem from the interaction of equations placed in different components.

As a consequence, stiff solvers are usually needed, the choice usually falling onto DASSL (for multi-step algorithms) and on Radau IIa (for single-step algorithms), which implement sophisticated step-size and order adaptation with error control, as well as root-finding algorithms for state-event detection.

When explicit solvers are required (e.g., for real-time simulation applications) it is sometimes possible to carefully build a modular model so that stiffness is avoided, but this is not the standard way people build object-oriented models in most cases, and people usually take for granted that stiffness will be handled by the solver.

In the early days of Modelica tool development, the largest and most challenging object-oriented models were multi-body systems, for which the above-sketched process can be extremely effective.

Consider, for example, the well-known 6 d.o.f. robot model of the Modelica.MultiBody.Examples library: a system which is originally described by 1766 non-trivial DAEs is reduced to an ODE system having only 36 states, whose derivatives can be computed by forward assignments, except for one 6x6 linear system. Furthermore, the resulting ODE system is dense and very strongly coupled, as a change in torque at one joint influences the acceleration of all the robot's links, due to kinematic constraints. The choice of a general-purpose dense ODE solver is perfectly adequate in this case. Even more dramatic is the case of the EngineV6 model, which starts from 2083 non-trivial DAEs and ends up with a 4-th order ODE system.

Probably due to the success in these very demanding applications, this standard work flow has not changed much over the years, and is still the state of the art as of today. However, there are significant problems when following this approach with several categories of models, some of which are rapidly gaining significance.

For example, it is well-known among control practitioners that the simulation of the transient of a Modelica system model including a digital controller can be orders of magnitude slower than the simulation of the model with the corresponding continuous-time one. As a consequence, people often delay the simulation of the actual closed-loop behaviour with the digital controller until late in the project, even though some potentially critical control functionality (e.g., anti-wind-up logic) cannot be easily and accurately reproduced in a continuous-time framework. Also, when finally switching to digital control, they might resort to fixed-time-step simulation, which do not give any guarantee of precision, in order to keep the simulation time within acceptable limits.

Another field of growing importance is the simulation of large networked systems with decentralized control. One notable example is that of smart grids, where multiple producers and consumers of electrical and also possibly thermal energy cooperate to the goals of stable network behaviour, satisfaction of all the load requests, and system optimality. Another interesting example is the one of self-driving cars on highways. These systems can easily encompass hundreds or thousands of individual agents, some of which might be inactive or dormant for long periods of time, as well as multi-domain physical phenomena that span widely different time-scales.

The design of the control strategy for such large-scale systems is usually based on hierarchical approaches, using cascaded control strategies and abstracting low-level behaviour within higher levels. However, at some point in the design cycle it becomes important to verify the system performance by taking into account the detailed physical behaviour, in particular to test how the system

reacts to borderline or anomalous conditions. For example, what if a power generation unit cannot keep up with the required load ramp rates, due to limitations in the energy conversion process? What if a self-driving car brakes too hard on slippery ground and loses traction in a rush-hour traffic scenario? Today's Modelica tools are clearly inadequate to handle the simulation of such large systems, because their standard work flow does not scale up well with the system size.

The goals of this paper are thus to point out the fundamental limitations of the current approach that hinder the use of Modelica tools in these areas, to highlight some recent relevant research developments going in the right directions, to make concrete proposals for further research, and finally to urge the Modelica community to undertake a more systematic and aggressive strategy to make the object-oriented simulation of such systems easy and efficient for tomorrow's system designers.

This is the outline of the paper: in Section 2, the issues of current state-of-the-art Modelica solvers with large-scale models are reviewed; Section 3 introduces a library that is intended to collect a wide array of benchmark of scalable (very) large Modelica models to support the development and testing of innovative methods and algorithms; Section 4 reviews some promising research trends to address the challenges of efficient simulation of large-scale models. Finally, Section 5 closes the paper with some concluding remarks.

2 Issues with State-of-the-Art Solvers and Large Models

In this section, the limitations of the standard work flow presented in the Introduction when dealing with large-scale models are discussed.

2.1 Localized Interaction is Not Exploited

Object-oriented system models are built in a modular way by the hierarchical composition of components and sub-systems via causal and a-causal connectors. The a-causal connection paradigm makes it possible to propagate instantaneous constraints (i.e., algebraic equations) through large portions of the system, with the consequence that the ODE $dx/dt = f(x,t)$ obtained after causalization is tightly coupled and has a dense Jacobian $\partial f/\partial x$ with comparably few non-zero terms.

In practice, however, this almost only happens in the case of multi-body systems, where the Jacobian corresponding to the states of kinematic chains turns out to be dense. In most other cases, the interaction between sub-systems is based on flows that depend only on nearby states. Although the a-causal modelling paradigm allows for algebraic constraints resulting in a tight coupling between the state derivatives across components, these constraints are usually confined to small portions of the sys-

tem. In other words, the derivative of any given state variable only depends on the values of a few neighbouring states, which means that each row of the Jacobian $\partial f/\partial x$ only has a few non-zero elements.

As a consequence, the Jacobian $\partial f/\partial x$ has a high degree of sparsity. Even more importantly, the number of non-zero element on each row is an invariant property of the system structure. Therefore, as the number of states N grows, e.g., because more and more individual agents or sub-systems are added to it, the number of non-zero terms only grows as $O(N)$, not as $O(N^2)$, as it is the case with tightly coupled systems.

Apparently, this basic fact is still not exploited by Modelica tool developers, which still employ dense solvers as the mainstream option. This has two severe consequences. The first is that the workload of Hessian inversion in implicit solvers, which grows as $O(N^3)$, eventually becomes the bottleneck for large enough systems, as there are no other tasks in the simulation process whose complexity grows at this rate. Probably, this is currently masked by the fact that other tasks become infeasible earlier as the size grows, before this break-even point is ever reached, but as other tasks are streamlined, this will become all-important.

The other consequence is that the memory allocated to store the values of the Jacobian $\frac{\partial f}{\partial x}$ and of the factored Hessian $(h\frac{\partial f}{\partial x} - I)^{-1}$ matrices for the solver can become unnecessarily quite large, triggering lots of cache memory misses that can severely degrade the simulation speed. For example, a system with 1000 state variables requires 16 Mbytes to store these two matrices in double-precision floating-point arithmetics; this is already well beyond above the size of the on-CPU cache in modern processors. A system with 10000 states requires 1.6 Gbytes of storage just for that purpose, which is totally unreasonable, as most of that space is occupied by zeros.

2.2 Localized Activity is Not Exploited

In many large-scale systems, local phenomena may occur within one sub-system or agent, that require short time steps for an accurate description, but on the other hand have negligible influence over the other sub-systems or agents on the time span of such short steps.

For example, consider the model of an urban district heating system. When a local temperature controller is switched on or off, or when a window is opened in one heated unit, relatively fast transients are triggered that involve local state variables, but have a negligible influence on the temperature of the water in the distribution system over that short time span, due to its large heat capacity. As a consequence, they also have a negligible effect on the other heated units. In order to describe the fast local transients within the specified accuracy, standard ODE solvers will reduce the time step length and

compute several time steps within a short time interval.

This is very inefficient when the system is very large, (say, 100 or 1000 heated units), since the short time steps span the *entire* system, requiring the computation of a very large derivative vector and of an extremely large Jacobian matrix, as well as the inversion of an extremely large Hessian matrix. This tremendous effort is in fact useless, as all other state variable will hardly change at all during these short steps.

2.3 Systems with Activity on Widely Different Time Scales Are Penalized

In many cases, multi-domain systems are characterized by physical phenomena taking place over widely different time scales. For example, power plant models built by connecting a boiler-turbine model, a synchronous electrical generator, and a transmission line to a network strong point, are characterized by slower thermal dynamic phenomena, taking place over time scales from a few seconds to a few hundred seconds, and faster electrical phenomena taking place over time scales from 10 to 100 milliseconds.

When transients take place in the faster sub-system(s), standard ODE solvers reduce the system-wide time step length to very small values, causing the wasteful re-computation of the slower thermal states, which hardly change at all across those time steps. On top of that, if accurate equations of state are used to describe the fluid properties, these unnecessary computations involve those equations, leading to an enormous and useless overhead. Once again, the slow-down factor gets bigger as the size of the system (i.e., the number of its states) increases.

2.4 Localized Influence of Events and Discontinuities is Not Exploited

When hybrid systems and events are involved, the situation illustrated in the previous sub-section gets even worse. The standard approach described in Section C of the Modelica Language Specification (The Modelica Association, 2014) requires that every time an event is triggered, the integration of the continuous-time ODE is halted at the event instant, the event is processed, global event iteration (involving the entire system!) is performed until convergence, and finally the simulation is restarted, usually with a very short time step because the discontinuities triggered by the event usually cause fast changes in some state variables, requiring small enough steps to stay within the allowed error tolerance.

Although this approach gives theoretical guarantees of accuracy in the numerical solution, it quickly becomes prohibitively expensive for all but the simplest systems.

For example, consider the model of an innovative energy conversion system for distributed generation, where

a waste heat recovery unit, powered by an Organic Rankine Cycle (ORC) engine, drives an electrical generator which is connected to the grid by means of a switched AC/AC converter. The ORC system has time constants from a few seconds to a few tens of seconds, and is characterized by a very high CPU load to compute its state derivatives, as complex equations of state are needed to describe the fluid properties. It might also feature a digital controller model with a periodic sampling time around 500 ms or so. On the other hand, the switched converter model triggers state events whenever currents or voltages on each of the three phases of both sides cross certain thresholds. Since a small ORC turbine can easily exceed 60000 rpm rotational speed and there are multiple events for each turn on each phase, the average frequency of events may easily exceed 10000 per second.

It is clear that recomputing the entire state derivative record, which requires computing all the fluid properties around the circuit, at this kind of rate can make the simulation four or more orders of magnitude slower than necessary, which basically means this kind of simulations is currently infeasible with state-of-the-art tools. On the other hand, this state of affairs is by no means necessary: it is easily understood that the effect of any single switching on the turbine rotational speed is negligible, due to its comparatively large inertia. Since the turbine is the only interface between the electrical part and the thermal part, there is obviously no need at all of recomputing the thermal states 10000 times per second or more, in order to achieve an accurate simulation.

2.5 Systems with large-scale algebraic constraints are not considered

At the other end of the spectrum, there are interesting system models characterized by very large systems of algebraic equations. In the Modelica world, a model with one or more such systems is often considered evil, or at least the result of inappropriate modelling practices, and the common wisdom calls for adding a few more states to the model in order to break them down to smaller systems. However, this is not always appropriate.

Consider the study of power generation and transmission systems, which has recently gained interest in the Modelica community, see Vanfretti et al. (2014). The models used to assess the network stability consider the dynamic phenomena taking place in the power generators, while neglecting the much faster electrical phenomena in the transmission network, which is described by algebraic equations (dynamic phasors). Nation-wide or continent-wide models can thus easily contain thousands of state variables, as well as one, very big algebraic system of a hundred thousands or more linear equations. The key factor here is that this system will be extremely sparse, thanks to the transmission network topology.

State-of-the-art tools try to cope with this system using tearing, which is prohibitively expensive at this scale.

2.6 Repetitive Structures are Not Exploited

Large-scale models usually involve repetitive structures, such as arrays of variables or models, and for-loops in equation sections. Even if for-loops are not used, it might be the case that the same model is instantiated a very high number of times. For instance, a model of a digital circuits might contain a very large number of NAND gates; a 64 bit adder might be built hierarchically by assembling 4-bit and 16-bit adders.

As mentioned in the Introduction, the mainstream approach of state-of-the-art tools is to flatten the entire model all the way down to scalar variables and equations, then analyse the structural properties of the system of equations and generate the code to compute the state derivatives. If the model is very large and has a lot of repetitive structures, the analysis phase (which is part of the compilation process) might require a very large amount of time, which could be spared if the analysis is carried out at the array level.

Also, following the standard approach, separate code is generated to solve each equation in the DAE, so that, in the case of repetitive models, the code has a large number of duplications, i.e., chunks of code that carry out exactly the same operation, albeit on different chunks of data. This can lead to unnecessarily high memory allocation, which brings in the previously discussed overhead due to off-cache memory access and cache misses. If the output of the Modelica tool is C code that needs to be compiled into executable code, very large source code files might also cause the C compiler to fail, or at least to become very slow, particularly if optimized code is generated.

3 The ScalableTestSuite Library

The assessment of the performance of existing tools when dealing with large scale systems, as well as the testing of innovative algorithms and solution strategies, calls for a library of benchmark cases. In the author's opinion, the requirements of this library are

- The size of the model should be easily selected by setting one (or more) integer parameters, while meaningful values of all other physical parameters should be automatically set by the model.
- The models should stress all the aspects mentioned in Section 2, either one at a time, or possibly also in a combined fashion.
- The models should be physically meaningful and representative of real-life modelling problems.
- The library should be self-contained and only depend on the Modelica Standard Library (MSL) to ensure maximum portability.
- At least some models should be defined as plain equations in a single Modelica class, so that they

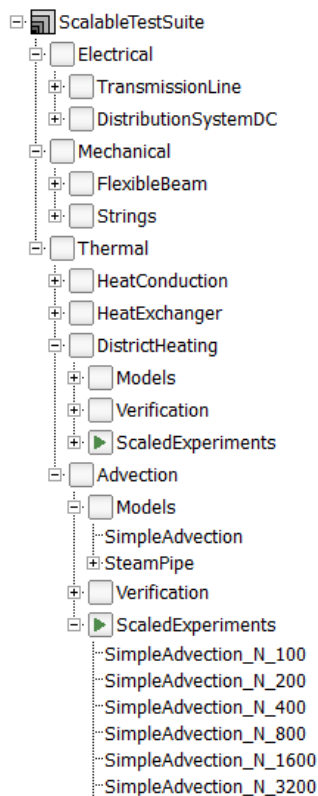


Figure 1. Structure of the ScalableTestSuite library.

might also be tried out easily with other simulation tools that do not support Modelica.

- At least some models should be defined both by plain equations and by modular descriptions, to assess how efficiently the tool can handle the overhead of a modular, object-oriented description of the model.
- The library should be widely advertised among tool developers and researchers, be open to contributions, and eventually become the accepted reference benchmark for the community.

A related work, the Modelimark test suite, was presented at the 2011 Modelica conference by Frenkel et al. (2011). The main goal in that case was the same of the library presented in this paper, i.e., provide scalable test models for tool benchmarking.

The result of that work is a Python software which can automatically generate Modelica code of models with scalable complexity. This approach is actually quite powerful and gives a lot of flexibility in terms of what can actually be tested; on the other hand, it makes the development, deployment and maintenance of the test suite more complex than a plain Modelica package containing the direct definitions of the benchmarks, in particular if many contributors are expected.

Also, the focus of that work is mainly (though not exclusively) aimed at evaluating the compiler performance,

i.e. how much time is needed to obtain the executable simulation code from the Modelica source, while the present work is more focused on the solver performance, though obviously the compiler performance can be evaluated as well. Finally, compared to the Modelimark test suite, the present work puts more emphasis on testing the simulation performance on physically meaningful models, which are representative of some class of real-life modelling problem.

At the 2014 EOOLT workshop in Berlin, after the discussion following the presentation of the paper (Ranade and Casella, 2014), the author decided to start working on his library with a master thesis project. Version 1.0 of the library, released on May 11th, 2015, is the end result of Kaan Sezginer's master's thesis (Sezginer, 2015). The library, which is open source and hosted on GitHub (Sezginer and Casella, 2015) is under continuous development and has already grown since then.

As of the time of this writing, the library contains 16 different test models, belonging to the electrical, mechanical, and thermal domains. The size can be set by suitable integer parameters - the library already contains the definition of test cases of size growing as the powers of 2, complete with experiment annotation, so that the benchmarks can be run by just checking out the library from the repository and compiling any of those classes.

The structure of the library is shown in Figure 1. More specifically, the following models are currently included:

- Electrical transmission line, directly modelled by equations or by connection of MSL components
- DC distribution networks, modelled by MSL components
- Flexible cantilevered beam, modelled by connection of MSL multi-body components or by the finite element method, taken from Schiavo et al. (2006)
- String suspended in a gravitational field, modelled by connection of MSL multi-body components
- One-dimensional heat conduction, with two different types of boundary conditions, directly modelled by equations or by connection of MSL components
- One-dimensional heat exchanger, in co-current and counter-current configuration, assuming constant fluid density and constant fluid heat capacity
- Models of 1D thermal advection, one assuming constant density and heat capacity, the other using the detailed IF97 model of steam and including compressibility effects
- Models of a district heating (Ranade and Casella, 2014) and of distributed cooling system (Floros et al., 2014). The former is a continuous time model, using nonlinear (and very stiff) systems with

bifurcations to model the local on-off temperature controllers; the latter employs events for the same purpose.

The models have been verified against known analytical solutions, whenever available. Each model stresses one or more of the aspects discussed in Section 2.

All the models in the library except the ones built with the MultiBody library have a high degree of sparsity, see Table 1 for some example values. As already anticipated in Section 2.1, the density of the Jacobian for the multi-body models does not depend on the size and is about 50%, making a dense solver perfectly adequate to handle them. Conversely, for all other models the number of non-zero elements grows as $O(N)$, so that the density of the Jacobian is inversely proportional to the system size and already much below 1% even for moderately large models with about 1000 state variables.

The district heating model is characterized by a strongly localized action: the on-off transitions of the local temperature controllers require many time steps to be computed accurately, but take place in much less than one second, during which the temperature of all other units do not change significantly; due to slightly different heat capacity parameters of the different units, all the transitions take place asynchronously, so that each transition involves one unit at a time.

The distributed cooling system model combines the feature of the previous model with events having only a local influence on the corresponding unit temperature.

The transmission line models also show localized action, as the simulated transient correspond to one sharp voltage and current wave crossing the transmission line once; consequently, each individual voltage has a sharp transition only when the wave passes through it, and is practically constant during the rest of the time.

The DC distribution system models allow to experiment with test cases featuring large sparse systems of linear algebraic equations.

As to repetitive structures, most examples are based on arrays of parametric size of variables and/or models, and make use of for loops in the equation section, so they can also be used for testing the ability of the compiler to cope with these structures efficiently. In the case of the DC distribution system, Modelica code is also provided that automatically generates the code of large system models with explicit declarations of individual components and connections, in order to test the ability of compilers to factor out common code also in this case. Some examples of automatically generated code are already included in the library.

More models with events, as well as multi-physics models with widely different time scales, will be added in the near future, possibly before the writing of the final version of this paper.

4 Research Trends for the Efficient Simulation of Large-Scale Models

This section points out some recent research trends that might improve the performance of Modelica tool dramatically when dealing with large-scale models. The aim is to encourage the community at investing more in this direction and bring these advanced methodologies in the mainstream as standard options for tomorrow's Modelica tools, so that they can keep up with the challenges posed by large-scale system models.

4.1 Sparse Solvers

A literature search on the topic of using sparse ODE solvers for the simulation of Modelica models found surprisingly little relevant results. To the author's knowledge, the only really relevant reference is a Modelica Conference paper (Link et al., 2009), where the authors advocated the use of sparse DAE solvers to improve the performance of power plant simulation in Modelica. Apparently, they have not been listened to so far.

To further motivate the analysis carried out in Section 2.1, Table 2 reports the simulation results of the SimpleAdvection model from the ScalableTestSuite library. The model is linear and the causalization can be performed exclusively by forward assignments, so that the Hessian inversion quickly becomes the performance bottleneck. Dymola 2015 FD01 has been used to run the test on a laptop with an Intel i5-4200U CPU and 8 GB of RAM, using DASSL as the ODE solver. Similar results have been obtained using OpenModelica on the same machine, with the same ODE solver.

The simulation time scales up as $O(N^{2.6})$ and it is really hard to believe that much better results couldn't be obtained with the sparse matrix version of DASSL. Also note the disproportionate amount of memory allocated by the simulation process (over 1 GByte for the largest model), mostly to accommodate the Jacobian and Hessian matrix values. Finally, note that in most scientific computing circles, a dynamic model with 12800 state variables is considered a small one, not a very large one.

The other application for sparse solvers is in models with large linear algebraic systems of equations. For very large systems, tearing takes too much time during code generation and leads to a set of residual equations which is still very sparse. Sparse numerical solvers should instead be automatically selected and used in these cases.

4.2 Multi-Rate Algorithms

In all those cases showing localized activity and/or widely different time scales, multi-rate algorithms can improve the simulation performance dramatically and increasingly with the system size.

Multi-rate algorithms have been studied since the early 1960s, but never made it into the mainstream sim-

Table 1. Sparsity of some large-size models.

<i>Model</i>	<i># of states</i>	<i># of non-zero Jac. entries</i>	<i>Jac. density</i>
Transmission line 320 elements	642	1603	0.38%
Transmission line 640 elements	1282	3203	0.19%
Steam pipe advection 320 volumes	640	3194	0.78%
Steam pipe advection 640 volumes	1280	6394	0.39%
Heat exchanger 320 volumes	957	3505	0.38%
Heat exchanger 640 volumes	1917	7025	0.19%
String models 32 segments	66	2147	49.29%
String models 64 segments	130	8387	49.63%

Table 2. Simulation performance of the SimpleAdvection model.

<i>Model</i>	<i># of states</i>	<i>Simulation time [s]</i>	<i>Memory allocation [MB]</i>
Simple advection, 1600 nodes	1599	6.7	22
Simple advection, 3200 nodes	3199	30.8	84
Simple advection, 6400 nodes	6399	183	326
Simple advection, 12800 nodes	12799	979	1293

ulation tools. The basic idea behind them is that only a few *global* steps involving the entire system should be performed. If the error estimates after one such step exceed the set tolerance only for a sub-set of states (i.e., the fast ones, or the ones with localized activity), then the time step grid is refined for those states only (the *active* states), while interpolating the found result for the *latent* states, whose precision has already been achieved with the global step. This approach can also be applied recursively.

The end result of using multi-rate algorithms is that whenever localized activity and/or transients in a fast subsystem take place, refinement steps are taken which only involve the (small!) relevant portion of the system, avoiding useless computation of other state derivatives, and also only requiring the inversion of small Hessians, in case of implicit solvers. Of course the advantage of using these algorithms grows with the size of the system.

On-going work at Politecnico di Milano is aimed at developing a multi-rate version of the TR-BDF2 algorithm, which is particularly attractive as the coefficients to interpolate the latent state are a by-product of the solution process, so they don't need any additional overhead. Very promising results have already been obtained using a multi-rate version of Rosenbrock's algorithm, applied to the district heating model of the ScalableTestSuite library. In particular, while the simulation time using DASSL grows as $O(N^{2.6})$, the simulation time of the prototype versions of multi-rate algorithms written in Matlab grows in a range from $O(N^{1.3})$ to $O(N^{1.8})$. Although the performance should be evaluated on an efficient version of the code written in C, the shown trend is nevertheless pointing in the right direction.

Also on-going at the moment of writing this paper is work at Bielefeld University to interface such solver to the OpenModelica compiler.

Please refer to (Ranade and Casella, 2014) for a more in-depth discussion of multi-rate algorithms, references to relevant literature, and preliminary results. Refer to (Casella, 2015) for ideas on how to generate efficient code to support such algorithms starting from declarative DAE-based Modelica models.

4.3 Smart Multi-Rate Event Handling

As discussed in Section 2.4, a straightforward implementation of the algorithm sketched in Appendix C of the Modelica Specification guarantees the correctness of the simulation results, but can be extremely inefficient in the case of large systems with frequently spaced events.

Some ideas have already been explored in the literature. Sanz et al. (2014) discuss how to avoid useless event iterations when it can be established a priori that a further event iteration is not necessary; they also discuss how to limit the event iterations to the smallest necessary sub-set of equations. In (Höger, 2013), a somewhat similar analysis is carried out to avoid unnecessary function evaluation during root finding, when the exact time instant of a state event is being computed. In both cases, the advantage grows with the size of the system.

All adaptive step-size solvers with error control of order n assume that the right-hand side of the ODEs is n times continuously differentiable. Whenever an event is triggered in a Modelica model, some derivatives can change discontinuously, which violates the assumption of continuous differentiability. The standard approach is

to integrate the equations up to the event time, process the event determining the new initial conditions, then restart the integration from these initial conditions completely disregarding the previous results. As said above, this approach is safe but usually very inefficient.

A less naive solution strategy could be worth exploring. Assume that the variable v with discontinuous behaviour due to events only influences the derivative of one state variable x_1 , that in turn x_1 only influences the derivative of x_1 and x_2 , and those in turn only influence the derivative of x_1 , x_2 , and x_3 . It follows that x_1 is continuous (though not differentiable), x_2 is continuously differentiable, and x_3 is twice continuously differentiable. By assumption, the rest of the system is not influenced by v , x_1 , and x_2 , so its inputs are twice continuously differentiable despite the event. Consequently, a second-order error estimation algorithm would give reliable indications about the error on all the states other than x_1 , x_2 , and x_3 across time step including the event.

Assume now that during a global step the zero-crossing function of the event changes sign, but the error estimator remains below the tolerance for all these other states. In the context of a multi-rate algorithm, it could then be possible accept the global step without any event handling, and only process the event in a refinement step, involving only the small set of active variables $\{x_1, x_2, x_3\}$.

In other terms, assuming that there is enough low-pass action between the discontinuous variables and the bulk of the system states, it might not be necessary to stop the integration of the latter ones, but only to refine the solution of those states which are more directly affected by the discontinuity.

There are of course a lot of details to be worked out to implement this approach, but it is the author's opinion that the performance improvements could be dramatic for most large models of systems subject to digital control; even more so if there are different sub-systems with different time scales and different sampling rates of the corresponding controllers.

4.4 QSS Algorithms

Quantized State Systems methods (Cellier and Kofman, 2006) adopt an alternative strategy for the solution of ODE systems, i.e., instead of discretizing over time they discretize over the set of state values, thus turning ODE systems into Discrete Event Systems (DEVS). Second- and third-order accurate methods QSS2 and QSS3 have been developed, as well as a linearly implicit method LIQSS (Migoni et al., 2013), which can deal with stiff system, a key feature for generic Modelica models.

Two features of QSS algorithms are relevant in the context of this paper. The first is that these algorithms naturally exploit localized interaction, as discussed in Section 2.1, handling them very efficiently, as demonstrated by Floros et al. (2014). The second is that, as soon

as the continuous-time ODEs are turned into event-based systems, the handling of events blends in the framework seamlessly without any significant overhead. Thus, also the issue raised in Section 2.4 can be solved efficiently by these algorithms.

Experiments have been already made at handling Modelica models with QSS algorithms.

Floros et al. (2011) report a first attempt at implementing an interface to a PowerDEVS-based QSS solver in the OpenModelica back-end. Unfortunately, this is no longer supported by OpenModelica due to later back-end refactoring.

Later on, the same research group developed a stand-alone QSS solver with a interface based on a small subset of Modelica (μ -Modelica), and a back-end module for OpenModelica that generates μ -Modelica code from regular Modelica code, see (Bergero et al., 2012).

Although these early experiments have lead to interesting published results, to the author's knowledge, as of the time of writing of this paper there is still no mainstream, well-tested and maintained tool that can process generic Modelica models, possibly involving advanced language features, and successfully generate simulation code based on QSS. Further work is required in this direction to consolidate the above-mentioned results.

4.5 Exploiting repetitive structures

Interesting ideas have been published about methods to avoid flattening the models to the level of scalar equations and variable, exploiting for loops or hierarchical model structure, see e.g. (Zimmer, 2009; Höger, 2011; Arzt et al., 2014). These methods may lead to considerable savings in the memory footprint of the simulation executable, as well as to considerable speed-up in the compilation and structural analysis phases.

However, at the time of the writing of this paper, these methods have only been demonstrated by prototype implementations, but are still unavailable in mainstream Modelica tools. More development, implementation and testing work is required to make them standard features of state-of-the-art Modelica tools.

4.6 Exploiting parallel CPUs

Research work on the parallelization of simulation code generated from Modelica models started as early as Peter Aronsson's PhD work (Aronsson, 2006).

The field is now finally approaching maturity. Parallel simulation code generation recently became available as an advanced feature in Dymola 2015 FD01, using algorithms described by Elmqvist et al. (2014), so it has already passed the stage of prototype implementation and entered the stage of advanced feature in at least one Modelica tool.

Two prototype parallel code generation back-ends in OpenModelica are documented in the literature:

(Sjölund et al., 2013), mainly focusing on TLM-based partitioning, and (Walther et al., 2014), based on ideas from Casella (2013). A new infrastructure to implement parallelized code is currently being built in the same tool, as documented by Gebremedhin and Fritzson (2014). To the author’s knowledge, no other published information is available concerning other Modelica tools.

The author hopes that this kind of algorithms eventually becomes a standard feature that does not require special settings by end user - the tool should figure out autonomously what is the best configuration for the problem at hand, given the available hardware resources (i.e., number of cores), and the exploitation of parallelism should become transparent to the end user, as most other low-level details of symbolic and numerical processing.

All the above-mentioned works exploit the parallelism inherent in the causalization process. It was clearly pointed out in the most recent papers that the problem of clustering the atomic tasks into bigger ones is essential to avoid excessive task set-up and switching overhead, which could more than offset the gain obtained by parallelism. Some heuristics are needed to estimate the actual workload, which is one key ingredient of clustering algorithm. One may expect that these heuristics will become more mature and reliable in the next few years.

Another useful feature when dealing with implicit solvers is the parallelization of the computation of the Jacobian $\partial f/\partial x$. This is almost trivial to achieve in the cases when the Jacobian is computed numerically, and also trivially obtained when the Jacobian is computed symbolically, once the problem of computing $f(x)$ efficiently has been solved. Significant speed-ups could be obtained here, in particular for large models.

As a side remark, it is surprising that the performance test whose results are reported in the above-mentioned references are carried out with low-end computers with few parallel cores, such as laptops. It would be interesting to see what kind of speed-up factors can be obtained if high-end workstations with latest-generation with 20+ logical cores are employed. This will give some understanding on what will be possible with run-of-the-mill hardware within 3-6 years.

As a final remark, it is the author’s opinion that parallel computation strategies should be combined with appropriate techniques exploiting sparsity and locality of large-scale Modelica models, in order to obtain truly outstanding performance improvement. Brute-force speed-up by parallelization is not able by itself to offset the inefficiencies pointed out in Section 2, even when larger numbers of cores will eventually become available at low cost on standard workstations.

5 Conclusions

In the future, large-scale system-level models of smart grids and distributed cyber-physical systems will become

a strategic asset in the development of such systems. Although the Modelica language has a lot of potential in this field, current state-of-the-art Modelica tools employ methods and algorithms that suffer from fundamental limitations as the size of the system model increases, quickly leading to unsatisfactory performance even for moderately large models. In order to meet the challenges posed by large, hybrid, distributed models, fundamental advances are required in the integration methods and also in the structural analysis and compilation phases.

This paper tries to draw the attention of the Modelica community on this topic, highlighting some fundamental issues, pointing out promising research trends that have potential to solve them effectively, and urging tool developers to pursue the goal of efficient simulation of large-scale models with determination.

Finally, the paper introduces the ScalableTestSuite Modelica package, a library of scalable models for benchmarking existing tools and helping the development of innovative methods and algorithms to cope with large-scale Modelica models. Contributions are welcome to improve and expand the scope of this library, in order to make it the reference collection of benchmarks.

6 Acknowledgement

The author thanks his former master’s student Kaan Sezginer for his contribution in developing version 1.0 of the ScalableTestSuite model library.

References

- P. Aronsson. *Automatic Parallelization of Equation-Based Simulation Programs*. PhD thesis, Linköping University, Department of Computer and Information Science, 2006.
- Matthias Arzt, Volker Waurich, and Jörg Wensch. Towards utilizing repeating structures for constant time compilation of large Modelica models. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 35–38, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666207.
- F. Bergero, X. Floros, J. Fernandez, E. Kofman, and F. Cellier. Simulating modelica models with a stand-alone quantized state systems solver. In *Proceedings 9th International Modelica Conference*, pages 237–246–442, Munich, Germany, Sep. 2012. Modelica Association. doi:10.3384/ecp12076237.
- Francesco Casella. A strategy for parallel simulation of declarative object-oriented models of generalized physical networks. In Henrik Nilsson, editor, *Proceedings of the 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, pages 45–51, Nottingham, UK, Apr 19 2013. ISBN 978-91-7519-621-3. URL <http://www.ep.liu.se/ecp/084/006/ecp13084006.pdf>.

- Francesco Casella. Efficient computation of state derivatives for multi-rate integration of object-oriented models. In I. Troch F. Breitenecker, A. Kugi, editor, *Proceedings 8th Vienna International Conference on Mathematical Modelling*, pages 262–267, Vienna, Austria, Feb. 18–20 2015.
- F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer-Verlag, 2006.
- Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. Parallel model execution on many cores. In *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, Mar. 10-12 2014. The Modelica Association. doi:10.3384/ECP14096363.
- X. Floros, F. Bergero, F. Cellier, and E. Kofman. Automated simulation of modelica models with qss methods - the discontinuous case. In *Proceedings 8th International Modelica Conference*, pages 657–667, Dresden, Germany, Mar 20-22 2011. Modelica Association.
- Xenofon Floros, Federico Bergero, Nicola Ceriani, Francesco Casella, Ernesto Kofman, and François Cellier. Simulation of smart-grid models using quantization-based integration methods. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings 10th International Modelica Conference*, pages 787–797, Lund, Sweden, Mar 10-12 2014. The Modelica Association. ISBN 978-91-7519-380-9. doi:10.3384/ECP14096787.
- Jens Frenkel, Christian Schubert, Günter Kunze, Peter Fritzson, Martin Sjölund, and Adrian Pop. Towards a benchmark suite for Modelica compilers: Large models. In *Proceedings 8th International Modelica Conference*, pages 143–152, Dresden, Germany, Mar 20-22 2011. Modelica Association.
- Mahder Gebremedhin and Peter Fritzson. Automatic task based analysis and parallelization in the context of equation based languages. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 49–52, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666210.
- C. Höger. Separate compilation of causalized equations. In *Proceedings 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools.*, pages 113–120, Sep. 2011.
- Christoph Höger. Sparse causalisation of differential algebraic equations for efficient event detection. In *Proceedings of the 8th EUROSIM Congress on Modelling and Simulation*, pages 351–356, Washington, DC, USA, 2013.
- K. Link, H. Steuer, and A. Butter. Deficiencies of modelica and its simulation environments for large fluid systems. In *Proceedings 7th International Modelica Conference*, pages 341–344, Como, Italy, Sep. 20–22 2009. The Modelica Association. ISBN 978-91-7393-513-5. doi:DOI: 10.3384/ecp09430034.
- G. Migoni, M. Bartolotto, E. Kofman, and F. Cellier. Linearly implicit quantization-based integration methods for stiff ordinary differential equations. *Simulation Modelling Practice and Theory*, 35:118–136, 2013. doi:10.1016/j.simpat.2013.03.004.
- Akshay Ranade and Francesco Casella. Multi-rate integration algorithms: a path towards efficient simulation of object-oriented models of very large systems. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 79–82, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666214.
- Victorino Sanz, Alfonso Urquia, and Francesco Casella. Improving efficiency of hybrid system simulation in modelica. In David Broman and Peter Pepper, editors, *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 21–28, Berlin, Germany, Oct 10 2014. ACM. ISBN 978-1-4503-2953-8. doi:10.1145/2666202.2666205.
- Francesco Schiavo, Luca Viganó, and Gianni Ferretti. Object-oriented modelling of flexible beams. *Multibody System Dynamics*, 15(3):263–286, 2006.
- K. Sezginer and F. Casella. The ScalableTestSuite Modelica library, 2015. URL <https://github.com/casella/ScalableTestSuite>.
- Kaan Sezginer. A test suite of large scalable models for Modelica tool evaluation. Master’s thesis, Politecnico di Milano, April 2015. Supervisor: prof. F. Casella.
- Martin Sjölund, Mahder Gebremedhin, and Peter Fritzson. Parallelizing equation-based models for simulation on multi-core platforms by utilizing model structure. In Alain Darté, editor, *Proceedings of the 17th Workshop on Compilers for Parallel Computing*, July 2013.
- The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 1.0. Online, Sep 1997. URL http://www.modelica.org/news_items/documents/ModelicaSpec30.pdf.
- The Modelica Association. Modelica - A unified object-oriented language for physical systems modeling - Language specification version 3.3 revision 1. Online, Jul. 11 2014. URL <https://www.modelica.org/documents/ModelicaSpec33Revision1.pdf>.
- Luigi Vanfretti, Tetiana Bogodorova, and Maxime Baudette. A Modelica power system component library for model validation and parameter identification. In *Proceedings 10th International Modelica Conference*, pages 1195–1203, Lund, Sweden, Mar 10-12 2014. The Modelica Association. doi:10.3384/ECP140961195.
- Marcus Walther, Volker Waurich, Christian Schubert, and Ines Gubsch. Equation based parallelization of modelica models. In *Proceedings of the 10th International Modelica Conference*, Lund, Sweden, Mar. 10-12 2014. The Modelica Association. doi:10.3384/ECP140961213.
- D. Zimmer. Module-preserving compilation of modelica models. In *Proceedings 7th International Modelica Conference*, pages 880–889, Como, Italy, Sep. 20–22 2009. The Modelica Association. doi:10.3384/ecp09430028.

Developing Mathematical Models of Batteries in Modelica for Energy Storage Applications

Dr. Thanh-Son Dao Dr. Chad Schmitke

Maplesoft, Waterloo, Ontario, Canada, {tdao, cschmitke}@maplesoft.com

Abstract

In this paper, effective and systematic steps in the mathematical modeling of high-fidelity battery models for simulating energy storage systems (ESS) will be presented. Two approaches to battery modeling will be discussed in this article: (1) the equivalent electrical circuit approach, and (2) the electrochemical approach. The battery models discussed in this article are developed based on the Modelica Standard Library specification 3.2.1 and commercialized as part of the Battery Component Library in MapleSim® 2015.

Keywords: electrochemical, battery, Modelica, energy storage, electric vehicles, MapleSim

1 Introduction

Battery modeling is a challenging field that has been receiving a great amount of interest recently due to the great push from the portable electronic devices and the electric and hybrid electric vehicles (EV/HEV) industries. Despite of the differences in the power ranges and battery sizes in these applications, the two industries share a common goal: developing a new generation of batteries that allow devices to run for a longer period of time, while operating within a range that maximizes the battery's service life. In both of these areas, accurate and efficient battery modeling is vital to help maximize the performance of a device and its battery.

In this paper, we focus on the development of mathematical models of batteries, with an aim to produce high-fidelity battery models that are suitable for a wide range of applications. The battery models are implemented using the Modelica language, allowing them to be shared across multiple simulation platforms.

There have been several efforts in the past few years to commercialize Modelica-based battery component packages, including the battery libraries developed by AIT (Einhorn *et al.*, 2011) and Modelon (Gerl *et al.*, 2014; Modelon, 2015). Although some of the basic features of battery simulation such as battery voltage, state of charge (SOC), state of health (SOH), thermal effects, etc., are included in these packages, they face many limitations, which are mainly due to the equivalent electrical circuits approach used in

generating their battery models. Some of the limitations include:

- It's not possible to maintain the battery voltage accuracy over a wide range of applied current: This is mainly because of the simplicity of the electrical circuit representations which do not allow them to describe the battery physics accurately under different operating conditions. For EV/HEV applications, this is a big obstacle as the charge/discharge current in an EV/HEV varies greatly during the operation of the vehicle and it is crucial that a battery model can portray this accurately.
- The thermal effects are simplified and neglect the changes in the entropy of the reaction of the electrode materials during each phase of the battery's operations. This is the most important heat source responsible for the reversible heat generation caused by the chemical reactions in the battery. As a result, the battery voltage and SOH cannot be simulated correctly as they are directly dependent on the battery temperature, making the battery components un-useful for battery parameter estimation and other applications that are temperature-dependent and require accurate simulation results. This includes studies and testing related to battery life and battery management systems (BMS).
- The circuit-based models are usually not scalable and are accurate only for the narrow ranges of specified parameter values. Any changes in the battery parameters such as capacity, thermal, applied current, etc., would cause large distortions in the simulation results.

In this article, two modeling approaches, namely the equivalent circuit approach, and the electrochemical approach are used to develop the battery models in MapleSim. All of the modeling issues mentioned above are either remedied or improved in the battery models presented in this article, depending on the modeling approach. The equivalent circuit models are upgraded based from the work (Chen and Rincón-Mora, 2006). Several new features are added to the original representation, allowing the models to capture the thermal effects, degradation, and electrode

chemistries precisely. This approach is described in details in Section 2. The electrochemical battery models are based on the porous electrode foundation proposed by (Newman and Tiedeman, 1975; Doyle *et al.*, 1996). The symbolic and numerical techniques for handling the electrochemical battery equations and incorporating the equations into MapleSim are thoroughly discussed in Section 3. Section 4 and Section 5 discuss thermal effects and battery degradation, while Section 6 presents the structure of the Modelica implementation of the models in MapleSim. The battery library includes a parameter estimation worksheet programmed in Maple to allow the users to validate the battery parameters based on experiment measurements. Section 7 provides information about this, and examples outlining some key applications of battery models are presented in Section 8.

2 Equivalent electrical circuit models

Circuit-based models attempt to model the electrochemical physics of a battery using only electrical components. These component models can easily be incorporated into any system models, and are generally not computationally expensive.

Although the circuit models abstract away the electrochemical physics happening in the battery, circuit-based models are close to mapping the physics onto circuit elements.

The base structure for the equivalent-circuit models for lead-acid, nickel-metal hydride, and lithium-ion cells in MapleSim was developed based on the work of (Chen and Rincón-Mora, 2006). The internal structure of the model is shown in Figure 1. There are two main linear circuits in the diagram: (1) the RC circuit at the bottom models the instantaneous, short, and long-time responses of the battery; and (2) the large capacitor at the top models the battery capacity.

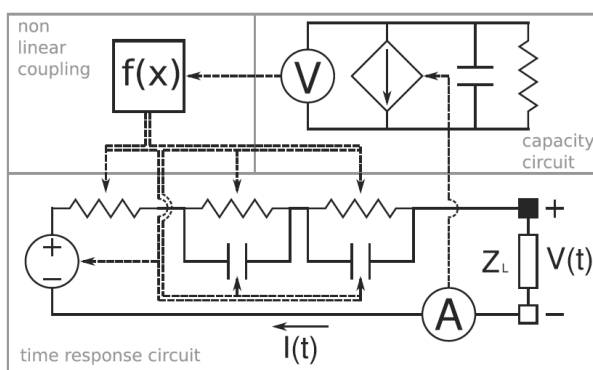


Figure 1. Schematic diagram of an equivalent circuit battery model.

The two circuits are non-linearly coupled using the following equations which describe the battery behaviors as the function of the battery SOC:

$$\begin{aligned}
 C_{cap} &= p_1 \\
 V_{oc} &= p_2 \exp(p_3 V_{soc}) + p_4 + p_5 V_{soc} + p_6 V_{soc}^2 \\
 &\quad + p_7 V_{soc}^3 \\
 R_{series} &= p_8 \exp(p_9 V_{soc}) + p_{10} \\
 R_{short} &= p_{11} \exp(p_{12} V_{soc}) + p_{13} \\
 C_{short} &= p_{14} \exp(p_{15} V_{soc}) + p_{16} \\
 R_{long} &= p_{17} \exp(p_{18} V_{soc}) + p_{19} \\
 C_{long} &= p_{20} \exp(p_{21} V_{soc}) + p_{22}
 \end{aligned} \tag{1}$$

For all materials, the open-circuit potential (OCP) equation is curve-fitted based on experiment measurements. As an example, Figure 2 shows the OCP versus SOC for some materials of a lithium-ion battery. Example curve-fitted equations for the OCPs of LiCoO_2 and graphite are shown in Equations (16) and (17) in Appendix A.

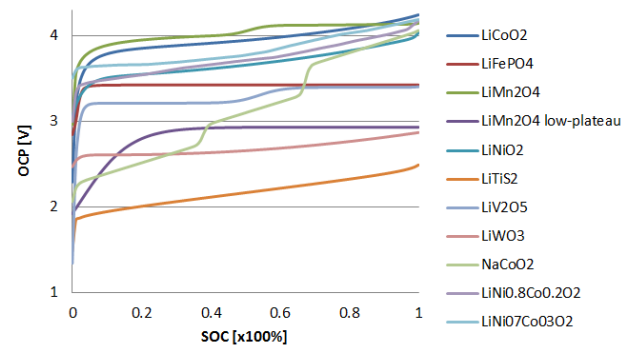


Figure 2. Open-circuit potentials for lithium-ion cathode materials.

3 Electrochemical models

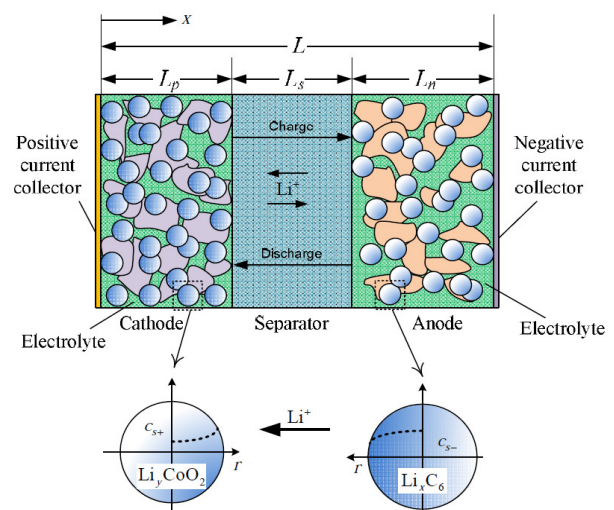


Figure 3. Anatomy of a lithium-ion cell (Dao *et al.*, 2012).

Electrochemical models explicitly represent the chemical processes that take place in the battery. These types of models describe the battery physics in great details, making them the most accurate of all the battery models.

3.1 Governing equations

Mathematical descriptions for electrochemical processes in most rechargeable cells are derived from the porous and concentrated solution theories proposed by (Newman and Tiedeman, 1975; Doyle *et al.*, 1996) which mathematically describe charge/discharge and species transport in the solid and electrolyte phases across a spatial cell structure. The porous electrode theory uses partial differential equations (PDE) to describe the electrochemical processes. In general, these PDEs are derived based on Fick's law of diffusion for the active material concentration, Ohm's law for electrical potential distributions, and the Nernst and Butler-Volmer equations. In this article, the PDEs governing a lithium-ion cell in 1D spatial structure will be used as an example (see Figure 3).

The PDEs that describe the changes in Li⁺ concentration in solid and liquid phases due to the gradient changes in the diffusive flow of Li⁺ ions are described by:

$$\frac{\partial c_s}{\partial t} = \frac{D_s}{r^2} \frac{\partial}{\partial r} \left(r^2 \frac{\partial c_s}{\partial r} \right) \quad (2)$$

and

$$\epsilon \frac{\partial c_e}{\partial t} = \frac{\partial}{\partial x} \left(D_{eff} \frac{\partial c_e}{\partial x} \right) + a(1 + t^+)j \quad (3)$$

Charge conservation in the solid phase of each electrode can be described by Ohm's law as

$$\sigma \frac{\partial^2 \Phi_s}{\partial x^2} = aFj \quad (4)$$

Combining Kirchhoff's law with Ohm's law in the electrolyte phase yields:

$$\begin{aligned} -\sigma_{eff} \frac{\partial \Phi_s}{\partial x} - \kappa_{eff} \frac{\partial \Phi_e}{\partial x} + \\ + \frac{2\kappa_{eff}RT}{F} (1 - t^+) \frac{\partial \ln c_e}{\partial x} = I \end{aligned} \quad (5)$$

The Butler-Volmer equation describing the relationships between the current density, concentrations, and over-potential is given by:

$$\begin{aligned} j = 2k(c_{s,max} - c_{s,surf})^{0.5} (c_{s,surf})^{0.5} c_e^{0.5} \\ \times \left[\exp\left(\frac{0.5F}{RT} \mu\right) - \exp\left(-\frac{0.5F}{RT} \mu\right) \right] \end{aligned} \quad (6)$$

These partial differential algebraic equations (PDAEs) are defined separately for each of the positive and negative electrode regions, and coupled with each other by the continuity in the boundary conditions. In total, 14 non-linear PDAEs are used to describe the behaviors of a lithium-ion cell. More details on battery equations can be found in (Newman *et al.*, 1975; Doyle *et al.*, 1996; Dao *et al.*, 2012; Seaman *et al.*, 2014)

3.2 Model reduction

The traditional methods for solving the battery PDAEs were mainly based on the classic finite difference techniques, which approximate continuous quantities

as being constant within discrete evenly-spaced intervals along the spatial axis, x , and approximate the derivatives based on a Taylor series expansion. The end result is a large set of hundreds or even thousands of linear differential equations (DEs) which makes solving the whole battery system in real-time extremely difficult.

The approach for handling the battery PDEs implemented in MapleSim is different. Our approach is to use the symbolic computation strength of Maple® 2015 to approximate the PDEs using Galerkin's method (Dao *et al.*, 2012; Seaman *et al.*, 2014) to produce fast simulating, yet accurate, models. The idea behind this method is to find the approximate numerical solution to a non-linear PDE using a set of orthogonal basis functions, and convert the PDE into a set of coupled ordinary differential equations (ODEs) based on temporal-spatial separation techniques. The resulting ODEs are small in size and are continuous, making the battery model they form more numerically stable and faster to solve.

Using the liquid phase concentration equation in Eq. (2) as an example, we start off with the spatial decomposition and allow time-varying coefficients as

$$c_{e,appx}(x, t) = \sum_{j=1}^N \mu_j(t) \alpha_j(x) \quad (7)$$

where N is the number of node points, $\alpha_j(x)$'s are the basis functions, and $\mu_j(t)$'s are the unknown functions of time to be solved for. The basis functions, $\alpha_j(x)$'s, must be orthogonal to ensure the resulting ODEs are linearly independent from each other.

Inserting approximate solution in (7) into the original PDE in (2) gives:

$$\begin{aligned} R_e(x, t) = \epsilon \sum_{i=1}^N \alpha_i(x) \frac{d\mu_i(t)}{dt} + \\ -D_{eff} \sum_i^N \frac{d^2 \alpha_i(x)}{dx^2} \mu_i(t) + a(1 - t^+)j \approx 0 \end{aligned} \quad (8)$$

This function is known as the residual. In Galerkin's method, we replace the condition that the residual should be approximately zero, with the condition that the residual should be orthogonal to the set of basis functions. The Galerkin techniques result in a set of ODEs which, when solved, give the approximate numerical solution, of the time-varying unknowns.

The reduced equations produced as the result of the Galerkin approach are then converted into Modelica to integrate with MapleSim using the Modelica component generation functionality in Maple.

4 Thermal effects

Modeling thermal effects in battery is crucial for many applications such as for testing and modeling the BMS, cooling system control, battery degradation, energy

consumption in automotive/mobile devices, etc. Currently, several commercial libraries such as those developed by AIT and Modelon, model battery thermal behaviors by only looking at the irreversible heat generation due to the ohmic thermal loss caused by the battery's internal resistance. This approach neglects the most important heat source – the reversible heat due to the chemical reactions in the battery electrodes.

Thermal effects are incorporated in both electrochemical and equivalent circuit battery components using the following lumped thermal equation:

$$\rho V C_p \frac{dT}{dt} = i(\mu_p - \mu_n) + iT \left(\frac{\partial U_p}{\partial T} - \frac{\partial U_n}{\partial T} \right) + i^2 R_{int} + h A_{surf} (T_a - T) \quad (9)$$

In the above equation, $\frac{\partial U_p}{\partial T}$ and $\frac{\partial U_n}{\partial T}$ are the entropy of reaction which are dependent on the electrode materials.

The entropy of reaction for each electrode chemistry is measured from experiments and integrated into the battery components using a lookup table. The entropy of reaction curves for some of the chemistries for a lithium-ion cell can be seen in Figure 4. MapleSim's lithium-ion cell component includes detailed electrochemical properties for 14 cathode chemistries and 3 anode chemistries, covering most of the commercially available lithium-ion electrode materials. The complete list of cathode and anode materials for lithium-ion cells supported by MapleSim 2015 is shown in Table 1 and Table 2 in Appendix B. The variety of the electrode materials allows the model to be used in different applications in both automotive and portable devices.

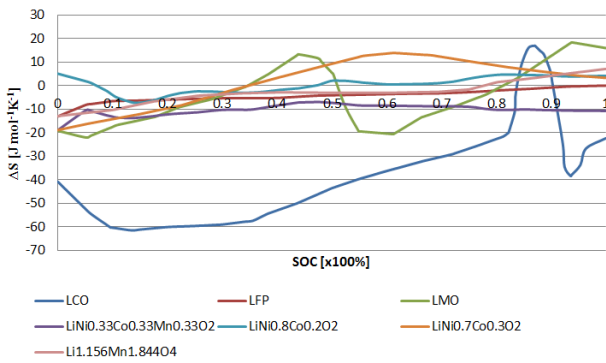


Figure 4. Entropy changes for lithium-ion cathode materials.

For all thermal models, the effects of cell temperature on the diffusion and ionic conductivity coefficients are modeled by Arrhenius' equation as:

$$D_s = D_{s,ref} \exp \left[\frac{E_{ds}}{R} \left(\frac{1}{T_{ref}} - \frac{1}{T} \right) \right] \quad (10)$$

$$D_{eff} = D_e \epsilon^{brugg} \exp \left[\frac{E_{de}}{R} \left(\frac{1}{T_{ref}} - \frac{1}{T} \right) \right] \quad (11)$$

$$\kappa_{eff} = \kappa \exp \left[\frac{E_k}{R} \left(\frac{1}{T_{ref}} - \frac{1}{T} \right) \right] \quad (12)$$

5 Capacity fade

The ability to model the degradation or capacity fade of rechargeable batteries during operation is critically important for predicting a battery's lifetime, and for designing and testing the BMS. In MapleSim, the battery life is modeled as the degradation caused by the formation of a solid-electrolyte interphase (SEI) layer in the negative electrode during charge (Pinson and Bazant, 2013). The following equation is used to describe the SEI thickness growth during charge:

$$\frac{ds}{dt} = \frac{k_{SEI} c_e M}{\left(1 + \frac{ks}{D_{diff}} \right) \rho_{SEI}} \quad (13)$$

Both the diffusion coefficient, D_{diff} , and the reaction rate, k_{SEI} , are temperature-dependent and can be described by Arrhenius' equation as:

$$D_{diff} = D_0 \exp \left(-\frac{E_a}{RT} \right) \quad (14)$$

$$k_{SEI} = A_e \exp \left(-\frac{E_a}{RT} \right) \quad (15)$$

The capacity fade is incorporated in both lithium-ion and NiMH cells for both electrochemical and equivalent circuit components based on the same degradation mechanism. The calculated battery SOH is dependent on the applied current, depth of discharge, temperature, and the cycling time. The degradation model also outputs the increase in the battery's internal resistance due to the formation of the SEI film.

$$R_{SEI} = \frac{S}{\kappa} \quad (16)$$

The parameters in the life model for lithium-ion cell have been estimated to closely fit the experiment measurements from a Lithium Iron Phosphate (LFP) found in (Liu *et al.*, 2010). A comparison shows that the differences between simulation results and experiment measurements are small, as shown in Figure 5.

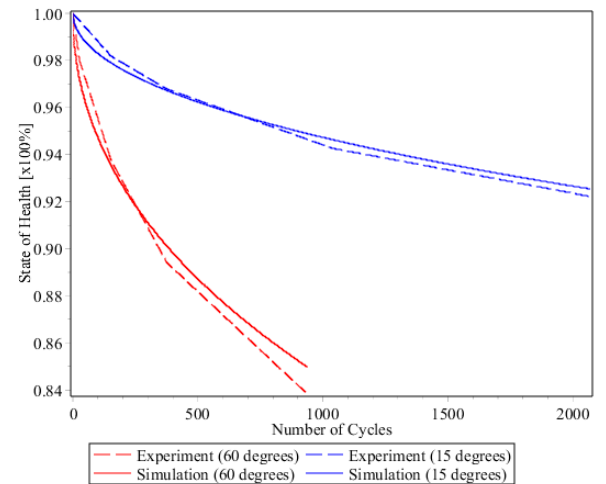


Figure 5. The life model (solid line) closely fits experiment data (dotted line) at 15°C and 60°C. Data from (Liu *et al.*, 2010) with LiFePO₄ electrode.

An example of the simulation results of a single lithium-ion cell being cycled for 20 hours at 80% Depth of Discharge (DOD) is shown in Figure 6.

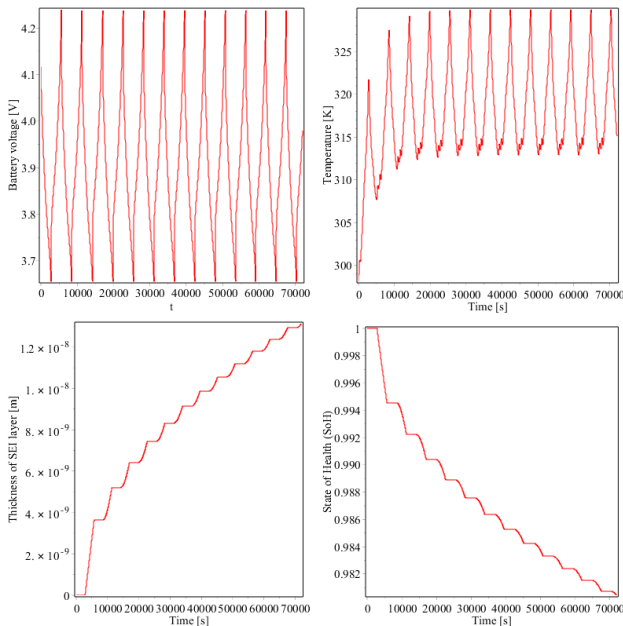


Figure 6. Degradation of LiCoO₂ electrode after cycling for 20 hours at 80% DOD. Plots show battery voltage, temperature, thickness of SEI layer, and state of health.

6 Modelica implementation

The battery code has been implemented based on the Modelica Standard Library specification 3.2.1. The order-reduced battery equations were converted directly into Modelica using Maple commands.

6.1 Single cell

Both the electrochemical and equivalent circuit cell components are based on the following structure:

```

model BatteryComponent "Descriptions"
public // main parameters
  parameter Real Ncell = 1 "Number of cells in series";
  parameter Boolean useCapacityInput = false "True means enable capacity input";
  parameter Real CA(unit = "A.h") = 1 "Battery capacity";
  RealInput Cin if useCapacityInput "Battery capacity input";
  parameter Boolean includeDegradation = false;
  // other electrochemical and thermal parameters
  ...
public // select electrode material
  parameter MaplesoftBattery.Selectors.Chemistry.Positive chem_pos;
  parameter MaplesoftBattery.Selectors.Chemistry.Negative chem_neg;
public // standard ports
  Voltage v;
  Current i;
  PositivePin p;
  NegativePin n;
public // optional heat models and input/output ports
  MaplesoftBattery.Selectors.HeatModel heatModel;
  RealInput Rint;
  RealOutput soc;

```

```

protected // protected parameters and internal variables
...
equation // main battery equations
...
end BatteryComponent;

```

The selectors for cathode and anode chemistries, `chem_pos` and `chem_neg`, are implemented as dropdown menus as shown in Figure 7, allowing the user to select the electrode materials easily. A suitable model for the electrochemical behaviors and thermal effects of the cell will be used, depending on the choice of the chemistry. Additionally, the user can incorporate custom data for electrode chemistries and thermal properties based on input signal or lookup tables.

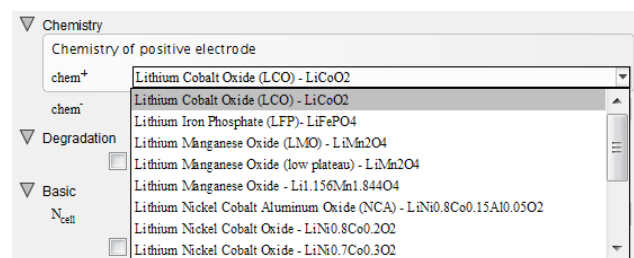


Figure 7. Lithium-ion cathode chemistry selector in MapleSim.

Thermal effects are also part of the battery components in MapleSim. There are three options for the thermal model: **isothermal**, **convection cooling**, and **external heat port**. The last option, **external heat port**, allows the other thermal components to be connected with the cell to simulate heat transfer between the cell and other parts of a system, such as heat transfer between the cells in a battery pack. If the **convection cooling** option is selected, the cell will exchange heat with an airflow through convection.

The battery degradation effects will be included in the simulation if the `useCapacityInput` option is selected. The battery model will output battery SOH in this case.

In addition to the main inputs/outputs, the user can turn on the input ports for the battery internal resistance and capacity, allowing these effects to be modeled based on a variable input or experiment measurements. Figure 8 shows a lithium-ion component with all the external ports turned on.

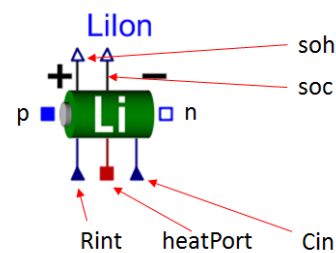


Figure 8. Ports of a battery component.

The battery components currently supported in MapleSim are shown in Figure 9.

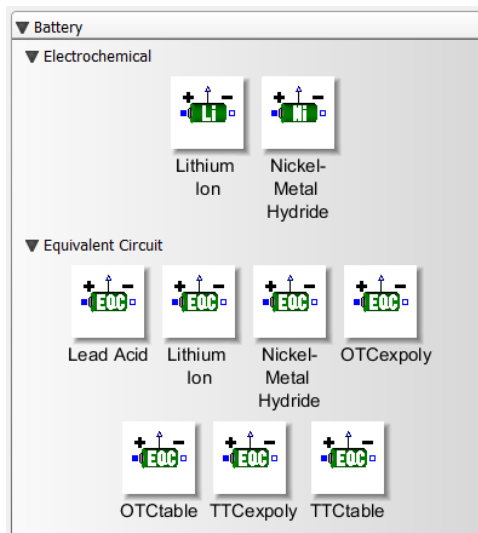


Figure 9. Battery components in MapleSim’s battery component library.

6.2 Stack

The user can wire the individual cells to create a stack model easily. This can be done using MapleSim’s advanced graphical user interface or based on standard Modelica descriptions. As an example, the Modelica script for n lithium-ion cells wired in series as shown in Figure 10 is written as follows:

```

model Main
public // cell definitions
  Electrochem.Lilon Cell1(chem_pos = ..., chem_neg = ...);
  Electrochem.Lilon Cell2(chem_pos = ..., chem_neg = ...);
  Electrochem.Lilon Cell3(chem_pos = ..., chem_neg = ...);
  ...
  Electrochem.Lilon Celln(chem_pos = ..., chem_neg = ...);
equation // cell connections
  connect(Cell1.p, ...);
  connect(..., Celln.n);
  ...
  connect(Cell3.p, Cell2.n);
  connect(Cell2.p, Cell1.n);
end Main;
    
```



Figure 10. Lithium-ion cells connected in series.

7 Battery parameter estimation

The Battery Component Library in MapleSim includes worksheets programmed in the Maple language, for estimating battery parameters, based on experiment measurements.

The global optimization technique of Differential Evolution Algorithm is implemented in the worksheet to effectively perform the search for multiple parameters and find the best possible solution. This algorithm is part of the Global Optimization Toolbox

in Maple and is powered by Optimus® technology from Noesis Solutions.

Figure 11 and Figure 12 show the parameter estimation results for a LiCoO₂ (LCO) lithium-ion battery based on two sets of test data.

Parameter	Current	Minimum	Maximum
CA	<input checked="" type="checkbox"/> 2.447	<input type="text" value=".1"/>	<input type="text" value="300"/>
Lp	<input checked="" type="checkbox"/> 0.7177e-4	<input type="text" value="0.10e-4"/>	<input type="text" value="0.200e-3"/>
Ln	<input checked="" type="checkbox"/> 0.8270e-4	<input type="text" value="0.10e-4"/>	<input type="text" value="0.200e-3"/>
ep	<input checked="" type="checkbox"/> .3846	<input type="text" value=".1"/>	<input type="text" value=".99"/>
en	<input checked="" type="checkbox"/> .3843	<input type="text" value=".1"/>	<input type="text" value=".99"/>
es	<input checked="" type="checkbox"/> .7254	<input type="text" value=".1"/>	<input type="text" value=".99"/>
efp	<input checked="" type="checkbox"/> 0.2565e-1	<input type="text" value="0.1e-1"/>	<input type="text" value=".2"/>
efn	<input checked="" type="checkbox"/> 0.5127e-1	<input type="text" value="0.1e-1"/>	<input type="text" value=".2"/>
Rsp	<input checked="" type="checkbox"/> 0.1518e-5	<input type="text" value="0.1e-5"/>	<input type="text" value="0.20e-4"/>
Rsn	<input checked="" type="checkbox"/> 0.9849e-5	<input type="text" value="0.1e-5"/>	<input type="text" value="0.20e-4"/>

Figure 11. Battery parameters used for curve fitting.

The plots in Figure 12 show that the simulation results closely resemble experiment data.

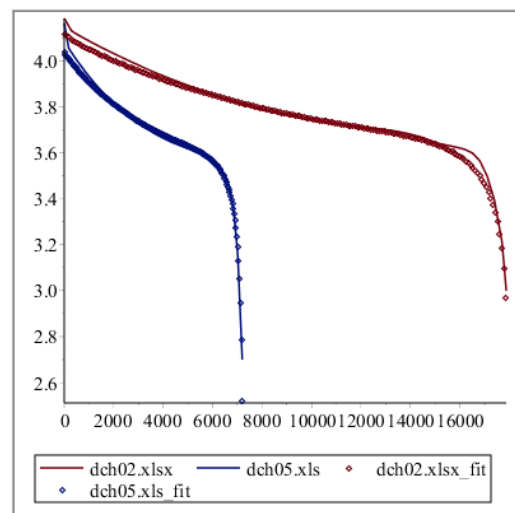


Figure 12. Simulation results (solid line) for battery parameter identification closely resemble experiment data (dotted line).

8 Applications

The structure and robustness of the battery components make them useful for many applications.

Example 1:

The example in Figure 13 shows an electric vehicle powered by a 30Ah Lithium Iron Phosphate (LFP) battery pack which has a maximum voltage of 390V when fully charged. The LFP cathode and Lithium Titanate (LTO) anode are chosen for their good thermal stability. The model also features an asynchronous induction motor. The combined efficiency of the motor and power electronics is modeled using an efficiency map, which allows the

simulated vehicle’s energy consumption to be realistic. The battery pack is air cooled, using thermal parameters defined through a parameter block. The lithium-ion component outputs key information about the battery’s operation such as SOC, temperature, voltage, and current values, as the vehicle follows the EPA highway drive cycle.

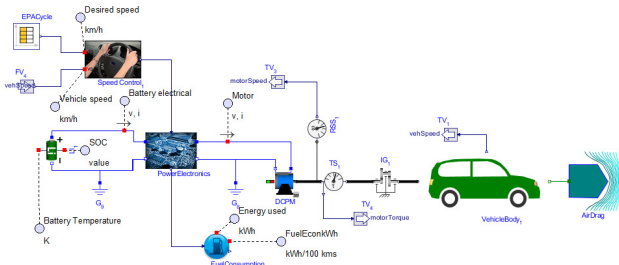


Figure 13. MapleSim model of an electric vehicle with an LFP lithium-ion battery.

The simulation results in Figure 14 show the vehicle speed and battery SOC. The controller does a good job in controlling the vehicle’s speed. Depending on the direction of the vehicle’s speed, the motor works in motoring mode. When the vehicle slows down, the motor becomes an electrical generator, which captures the brake energy to recharge the battery through regenerative braking.

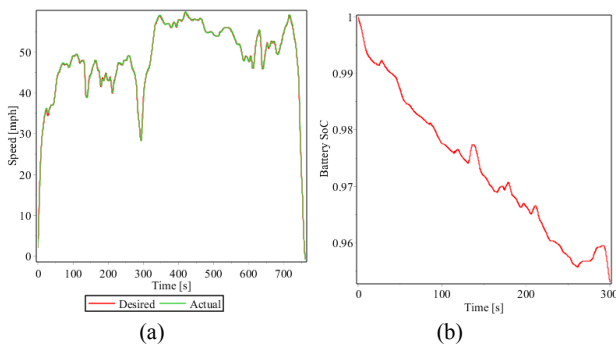


Figure 14. Vehicle speed (a) and battery SOC (b).

Example 2:

The following example (Figure 15) shows the simulation of the thermal exchange between lithium-ion cells connected in a series/parallel structure. The 6 cells used in the model exchange heat between the adjacent cells that are in contact with them through thermal conductivity.

The differences in cell temperatures result in an imbalance in the cell voltages (*i.e.*, some cells have lower voltages than the other cells), causing the currents through the series strings to vary (see Figure 16). In extreme cases, the “weak” cells (*i.e.*, cells having lower voltages) can become over-heated due to short-circuiting and have to be remedied using a balancing current. The stack voltage is shown in Figure 17.

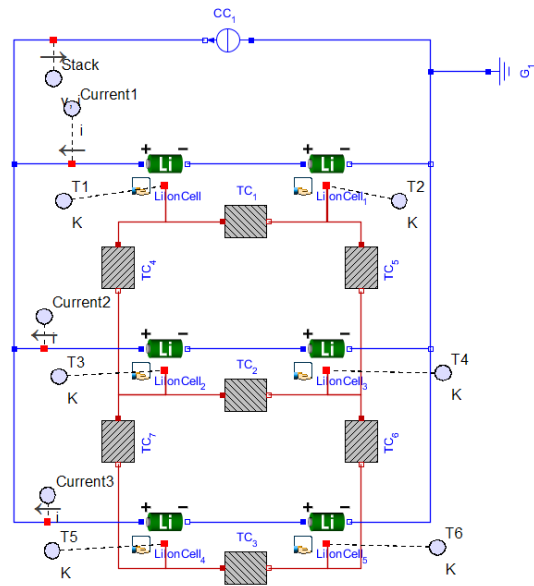


Figure 15. Thermal exchange in battery pack wired in series/parallel configuration.

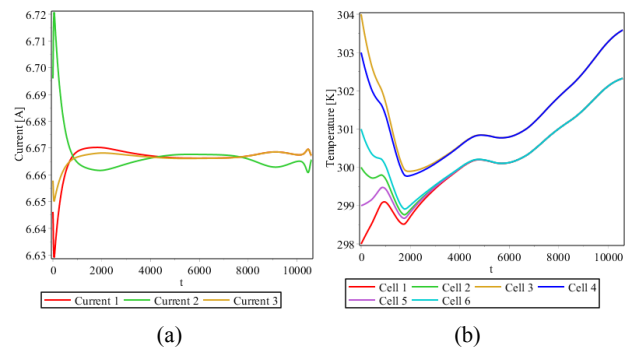


Figure 16. Current through each string of cells connected in series (a) and cell temperatures (b).

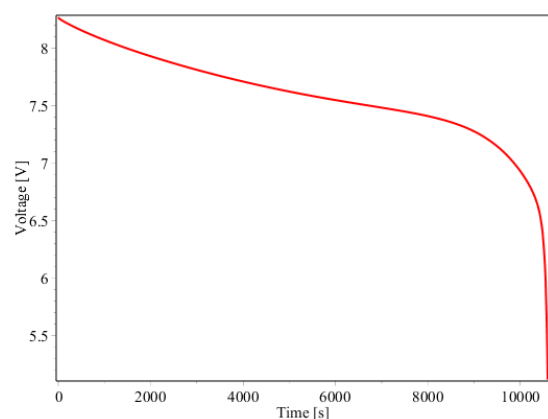


Figure 17. Stack voltage.

9 Conclusion

In this paper, we have shown two methods of generating battery model. While equivalent electrical models attempt to model the electrochemical physics of a battery using only electrical components, the

resulting battery component, which is computationally inexpensive to incorporate into system models, has many limitations.

Electrochemical models on the other hand, are the most accurate because they describe the physics of a battery by explicitly representing the chemical processes that take place within it. Starting with PDEs to describe these electrochemical processes, symbolic techniques are then applied to approximate the PDEs using Galerkin's method, to arrive at the final set of reduced equations.

Capacity fade is incorporated into both the equivalent circuit and electrochemical battery components, with simulation results showing that battery life achieved by the components is a close fit with experiment measurements. The battery components also incorporate thermal effects, and include Maple language-based worksheets for battery parameter estimation, based on experiment measurements.

The battery components were implemented based on the Modelica Standard Library specification 3.2.1, and commercialized as part of the Battery Component Library in MapleSim™. The battery library also comes with a parameters identification worksheet which ensures a high level of fidelity in the battery components, making them suitable for a wide range of applications.

References

- M. Chen and G. Rincón-Mora. Accurate Electrical Battery Model Capable of Predicting Runtime and I-V Performance. *IEEE Trans. On Energy Conversion*, 21(2):504-511, 2006.
- T.-S. Dao, C.P. Vyasarayani, and J. McPhee. Simplification and Order Reduction of Lithium-Ion Battery Model Based on Porous-Electrode Theory. *Journal of Power Sources*, 198:329-337, 2012.
- M. Doyle, J. Newman, C. Schmutz, and J.M. Tarascon. Comparison of Modeling Predictions with Experimental Data from Plastic Lithium Ion Cells. *Journal of the Electrochemical Society*, 143(6):1890-1903, 1996.
- M. Einhorn, F.V. Conte, C. Niklas, H. Popp, and J. Fleig. A Modelica Library for Simulation of Electric Energy Storages. *The 8th International Modelica Conference*, 2011.
- J. Gerl, L. Janczyk, I. Krueger, and N. Modrow. A Modelica Based Lithium Ion Battery Model. *The 10th International Modelica Conference*, 2014.
- P. Liu, J. Wang, J. Hicks-Garner, E. Sherman, S. Soukiazian, M. Verbrugge, H. Tatara, J. Musser, and P. Finamore. Aging Mechanisms of LiFePO₄ Batteries Deduced by Electrochemical and Structural Analysis. *Journal of the Electrochemical Society*, 157(4):A499-A507, 2010.
- J. Newman and W. Tiedeman. Porous-Electrode Theory with Battery Applications. *AIChE Journal*, 21(1):25-44, 1975.
- M.B. Pinson and M.Z. Bazant. Theory of SEI Formation in Rechargeable Batteries: Capacity Fade, Accelerated Aging, and Lifetime Prediction. *Journal of the Electrochemical Society*, 160(2):A243-A250, 2013.
- A. Seaman, T.-S. Dao, and J. McPhee. A Survey of Mathematics-Based Equivalent-Circuit and Electrochemical Battery Models for Hybrid and Electric Vehicle Simulation. *Journal of Power Sources*. 256:410-423, 2014.
- Modelon Battery Library [Web]:
<http://www.modelon.com/products/modelica-libraries/battery-library-release-information/>

List of symbols

a	specific surface area [m^2m^{-3}]
A_e	factor for reaction rate equation [m s^{-1}]
$brugg$	Bruggeman's coefficient
c_e	concentration in electrolyte phase [mol m^{-3}]
c_s	concentration in solid phase [mol m^{-3}]
$c_{s,max}$	maximum concentration in solid phase [mol m^{-3}]
$c_{s,surf}$	surface concentration in solid phase [mol m^{-3}]
D_0	diffusion coefficient at standard conditions [m^2s^{-1}]
D_s	diffusion coefficient [m^2s^{-1}]
$D_{s,ref}$	reference diffusion coefficient [m^2s^{-1}]
D_{eff}	electrolyte diffusion coefficient [m^2s^{-1}]
E_a	activation energy [J mol^{-1}]
E_{de}	activation energy for diffusion [J mol^{-1}]
E_{ds}	activation energy for diffusion [J mol^{-1}]
E_k	activation energy for ionic conductivity [J mol^{-1}]
I	applied current [A]
j	wall-flux of ions [$\text{mol m}^2\text{s}^{-1}$]
k	reaction rate constant [$\text{mol}(\text{mol}^{-1}\text{m}^3)^{3/2}$]
k_{SEI}	rate constant of SEI formation [m s^{-1}]
M	molar mass of SEI layer [kg mol^{-1}]
r	radius of intercalatin particle [m]
R	ideal gas constant [$\text{J K}^{-1}\text{mol}^{-1}$]
s	thickness of SEI layer [m]
t	time, [s]
t^+	transference number in the electrolyte
T	battery temperature [K]
T_{ref}	temperature at standard conditions [K]
x	main dimension across the cell sandwich [m]
α_i	basis function
ϵ	porosity of electrode
Φ_s	electrical potential in solide phase [V]
Φ_e	electrical potential in electrolyte phase [V]
κ	Specific conductivity coefficient [m A^{-1}]
κ_{eff}	effective ionic conductivity of electrolyte [S m^{-1}]
μ	over-potential [V]
μ_i	time-dependent variable of the i-th basis function
ρ_{SEI}	density of SEI layer [kg m^{-3}]
σ	electronic conductivity of in solid phase [S m^{-1}]

Appendix A

OCP equations for LiCoO₂ cathode and LiC₆ anode:

$$U_p = (-3.234 - 638.136\theta + 2387.637\theta^2 - 4027.7467\theta^3 + 4106.484\theta^4 - 2790.517\theta^5 + 1292.901\theta^6 - 401.8703962\theta^7 + 79.910\theta^8 - 9.1764\theta^9 + 0.463\theta^{10}) / (-1.411 - 160.70\theta + 604.048\theta^2 - 1008.0656\theta^3 + 1005.1301\theta^4 - 663.87\theta^5 + 299.134\theta^6 - 90.922\theta^7 + 17.8147\theta^8 - 2.032\theta^9 + 0.1025\theta^{10}) \quad (17)$$

$$U_p = 0.7222 + 0.1186\theta + 0.268\theta^{0.5} - \frac{0.20114}{\theta} + \frac{0.2403}{\theta^{1.5}} + 0.2808\exp(0.90 - 12.8265\theta) - 0.798\exp(0.3818\theta - 0.4108) \quad (18)$$

In both equations, θ indicates the battery SOC.

Appendix B

Table 1. List of cathode materials for Lithium-ion cells supported by MapleSim.

<i>Chemical composition</i>	<i>Common name, where given</i>
LiCoO ₂	LCO
LiFePO ₄	LFP
LiMn ₂ O ₄	LMO
LiMn ₂ O ₄ – low plateau	
Li _{1.156} Mn _{1.844} O ₄	
LiNi _{0.8} Co _{0.15} Al _{0.05} O ₂	NCA
LiNi _{0.8} Co _{0.2} O ₂	
LiNi _{0.7} Co _{0.3} O ₂	
LiNi _{0.33} Mn _{0.33} Co _{0.33} O ₂	NMC
LiNiO ₂	
LiTiS ₂	
LiV ₂ O ₅	
LiWO ₃	
NaCoO ₂	

Table 2. List of anode materials for Lithium-ion cells supported by MapleSim.

<i>Chemical composition</i>	<i>Common name, where given</i>
LiC ₆	Graphite
LiTiO ₂	
Li ₂ Ti ₅ O ₁₂	LTO

Average model of a synchronous half-bridge DC/DC converter considering losses and dynamics

Michael Winter¹ Sascha Moser¹ Stefan Schoenewolf¹ Julian Taube¹
Hans-Georg Herzog¹

¹Institute for Energy Conversion Technology, Technische Universitaet Muenchen, Germany,
michael.winter@tum.de

Abstract

Nowadays, power electronic systems play a major role in almost every large system. Due to the high switching frequencies, the simulation of these devices is computationally very expensive and not suitable for system simulation. Average models of these power electronic systems are needed to simulate the basic terminal characteristics of these devices without the need to simulate every switching operation. This paper describes a Modelica implementation of a synchronous half-bridge converter for the use in an automotive power net simulation as well as in real-time environments. The model takes into account the losses in the semiconductors as well as the dynamic behavior of the converter. For the parametrization of the model, only the switching frequency and some values from the datasheets of the used components are required. To validate the proposed model, an equivalent SPICE model is developed, serving as a reference model. The dynamic behavior of the two models is compared using step responses of the load current. The relative deviation of the model's output voltage compared to the SPICE simulation is less than 2%. Furthermore, also the energetic behavior was investigated, and it is shown that the proposed model provides good results for a wide operating area.

Keywords: power electronics, average model, DC/DC converter, losses, acausal modeling

1 Introduction

DC/DC converters are used to convert a DC input voltage into a DC output voltage with a higher, lower or inverted value. In the automotive power net, such power electronic circuits are used extensively. Almost every electronic control unit (ECU) has a DC/DC converter close to its terminals to the power net side in order to compensate voltage fluctuations and to supply the electronics with a constant

voltage. In addition, DC/DC converters are used to control DC motors, to couple the 48V level with the 12V power net or to stabilize the 12V power net with an additional energy storage (Ruf et al., 2012).

In today's product development processes, simulation is a very important step. With the help of simulation, development cycles can be shortened and costs can be reduced. When simulating power electronic systems, several engineering challenges have to be solved. During the development phase, the DC/DC converter is usually simulated with SPICE (Simulation Program with Integrated Circuit Emphasis) or a similar software. The behavior of the converter is primarily determined by the power semiconductors and passive components. The passive components can be easily modeled in SPICE and the power semiconductor manufacturers provide more or less accurate models for their products. The major advantage of a SPICE simulation is that the dynamic behavior and all loss mechanisms in a power electronic system are taken into account. The simulation is computationally quite intensive, however, the long computation times are not a big issue in this development phase as only short periods of time are considered and usually only one or a small number of converters are simulated simultaneously.

For system simulation however, these models are not suitable for two reasons. On the one hand, in contrast to Modelica, SPICE does not meet the requirements of multi-domain simulation. There are already approaches to translate SPICE models into the Modelica language (Majetta et al., 2011) and it also has been shown that Modelica is in principle suitable for simulation of power electronics (Glaser et al., 1995). None of these approaches solves the second challenge, that the calculation of the switching operations would slow down the system simulation, so it would be practically impossible to carry out longer simulations. Another approach is to build behavioral or loss models. The various loss mechanisms in power electronic converters are well understood and can be described by algebraic equations

(Giuliani et al., 2011; Gragger, 2011). However, this type of modeling neglects the dynamic behavior of the passive components.

2 Objectives and Approach

This paper describes an average model of a synchronous half-bridge converter in continuous conduction mode, considering the converter's dynamics and losses. In future work, the model will be used both in the Modelica-based automotive power net simulation environment (Ruf et al., 2013) as well as in real-time operation in an automotive power net test bench (Kohler et al., 2010). For parametrization of the model, only the switching frequency and the datasheet parameters of the power semiconductors, the inductance, and the capacitors are needed. The model is validated by means of an equivalent SPICE model.

3 The Synchronous Half-Bridge Converter

The converter model presented in this work is based on a synchronous half-bridge converter as shown in Figure 1. The fundamentals presented in this chapter are well known and documented in literature (Kazimierczuk, 2008; Erickson and Maksimović, 2001), as well as in application sheets provided by the semiconductor manufacturers (Vishay, 2002). The converter consists of a high-side switch S , a low-side switch \bar{S} , an inductor L , and capacitors at the input and output. Basically it is a standard buck or boost converter topology with the freewheeling diode replaced by a second switch in order to reduce conduction losses. Furthermore, the topology becomes bidirectional by this change where the voltage v_1 is always higher than the voltage v_2 . The direction of the energy flow can be controlled by the duty cycle D .

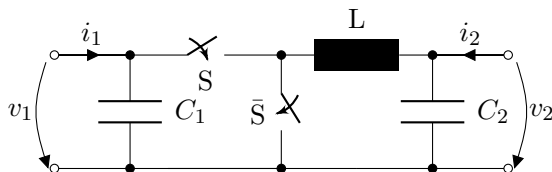


Figure 1. Topology of a synchronous half-bridge

3.1 General equations for the synchronous half-bridge

The two switches are controlled by complementary pulse width modulated signals with the switching

frequency f_S . The duty cycle D relates in the following always to the high-side switch S and is defined as follows:

$$D = \frac{t_{on}}{T_S} = \frac{t_{on}}{t_{on} + t_{off}} = f_S \cdot t_{on} \quad (1)$$

The transfer equations for a loss-less half-bridge containing no storing elements such as inductors or capacitors are the same as for an ideal DC-transformer:

$$v_2 = D \cdot v_1 \quad (2)$$

$$i_2 \cdot D = -i_1 \quad (3)$$

In order to obtain a transformer considering the ohmic and the switching losses, equation (2) is replaced by a power balance equation that interrelates input power, output power and power dissipation:

$$P_1 + P_2 - P_L = 0 \quad (4)$$

The load dependent losses of the half-bridge are composed of the losses of the high-side and the low-side switch. In each switch, conduction losses occur and depending on the mode, also switching losses.

$$P_L = P_{L,S,cond} + P_{L,S,sw} + P_{L,\bar{S},cond} + P_{L,\bar{S},sw} \quad (5)$$

The load dependent loss mechanisms in the semiconductors are presented separately in the following two subsections. For switching losses, the turn-on and turn-off times of the power semiconductors have a major impact. In this work, the simplifying assumption is made that both switches are the same and the driver circuit has an ideal behavior, providing similar rise and fall times t_{rise} and t_{fall} . Thus, switching times are simply a function of the total gate charge Q_g , the operating voltage of the gate-drive circuit U_{drive} and the sum of the gate driver's output resistance, the gate series resistance and the gate's input resistance, called $R_{g,total}$.

$$t_{rise} = t_{fall} = \frac{Q_g \cdot R_{g,total}}{U_{drive}} \quad (6)$$

In addition the continuous charging and discharging of the semiconductor's gates causes losses which have to be covered by the gate driver. These losses are a function of the total gate charge $Q_{g,total}$, the driving voltage V_{drive} and the switching frequency.

$$P_{L,Gate} = Q_{g,total} \cdot V_{drive} \cdot f_S \quad (7)$$

3.2 Loss equations for the synchronous half-bridge in buck mode

In buck mode the high-side switch is obligatory, so conduction and switching losses occur:

$$P_{L,S,cond} = R_{DS,on} \cdot i_2^2 \cdot D \quad (8)$$

$$P_{L,S,sw} = 0.5 \cdot v_1 \cdot (-i_2) \cdot (t_{fall} + t_{rise}) \cdot f_S \quad (9)$$

The low-side switch does not have to switch the current but supports the loss reduction in the commutation path. For simplicity it is assumed that losses through the conducting diode during the dead times and the reverse recovery effect can be neglected, so here only conduction losses occur:

$$P_{L,\bar{S},cond} = R_{DS,on} \cdot i_2^2 \cdot (1 - D) \quad (10)$$

$$P_{L,\bar{S},sw} = 0 \quad (11)$$

3.3 Loss equations for the synchronous half-bridge in boost mode

The low-side switch is the primary switch during boost mode operation, the losses are described by following equations:

$$P_{L,\bar{S},cond} = R_{DS,on} \cdot i_1^2 \cdot (1 - D) \quad (12)$$

$$P_{L,\bar{S},sw} = 0.5 \cdot v_2 \cdot (-i_1) \cdot (t_{fall} + t_{rise}) \cdot f_S \quad (13)$$

The high-side switch is part of the commutation path, so there are only conduction losses:

$$P_{L,S,cond} = R_{DS,on} \cdot i_1^2 \cdot \frac{(1 - D)}{D^2} \quad (14)$$

$$P_{L,S,sw} = 0 \quad (15)$$

4 Models

In this chapter, the proposed Modelica model of a synchronous half-bridge converter is presented. Then the equivalent model is implemented in SPICE and serves as a reference model in chapter 5.

4.1 Modelica implementation of the synchronous half-bridge converter

The implementation of the model of the proposed half-bridge converter is based on the averaged circuit model for a two-switch converter (Erickson and Maksimović, 2001). The model is obtained by taking the basic structure of the real converter including the inductance and capacitances. Then, the switches have to be replaced by an averaged switch model, as shown in Figure 2. The electrical connectors $p1$ and $n1$ provide the terminals for the high-voltage side, the connectors $p2$ and $n2$ for the low-voltage side, and the connectors p_a and n_a for the auxiliary power supply. The auxiliary power supply can optionally be charged with a constant power loss in order to take the power for the converter's control into account. The transfer function and the loss equations of the semiconductors of the synchronous half-bridge are implemented in a separate model named *AveragingHalfBridge*. This sub-model contains equations (3) - (7)

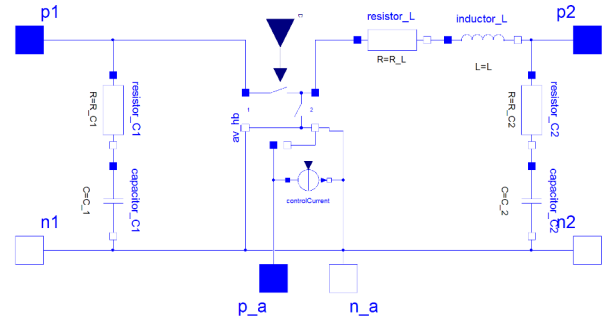


Figure 2. Model of the proposed synchronous half-bridge converter

and is able to swap between the two equation systems (8) - (11) for buck and (12) - (15) for boost mode, depending on the actual current direction. The *AveragingHalfBridge* is derived from the base class *Modelica.Electrical.Analog.Interfaces.TwoPort* of the *Modelica Standard Library* which also defines the directions of the currents and voltages of the equation system. The sub-model receives the actual duty cycle D of the half-bridge as a real input and is parametrized with the following values:

- The switching frequency f_S of the half-bridge.
- The on-resistance $R_{DS,on}$ and the total gate charge $Q_{g,total}$ of the MOSFET, both values can be taken from the MOSFET's datasheet.
- The operating voltage of the gate-drive circuit U_{drive} and total gate resistance $R_{g,total}$, determined by design and the MOSFET's and driver's parameters from the datasheet.

The dynamic behavior of the converter is determined by the input and output capacitors, the inductance and the respective series resistances of these components:

- The capacitance C_1 and the equivalent series resistance of the capacitor R_{C1} on the high-voltage side of the converter.
- The inductance L and the equivalent series resistance of the inductor R_L .
- The capacitance C_2 and the equivalent series resistance of the capacitor R_{C2} on the low-voltage side of the converter.

4.2 SPICE model as reference for validation

For the validation of the Modelica model, a SPICE model for comparative simulations is implemented which is shown in Figure 3. This model takes the switching behavior of the real converter into account. The passive components of the converter

like input and output capacitors and inductor are directly inserted from the SPICE library. These components are parametrized to the same component values as for the Modelica model in subsection 4.1, taking into account the ohmic resistance of the inductor and the equivalent series resistance of the capacitors. For the MOSFETs and anti-parallel freewheeling diodes SPICE models provided by the semiconductor manufacturers are used. The gate drivers are modeled by time varying voltage sources. An additional resistor corresponds to the gate driver's output resistance and gate series resistance whereas the gate's input resistance is provided by the model of the MOSFET. Dead time as well as rise and fall times of the gate driver are set by the voltage shape of the voltage sources. As all parameters and submodels of the SPICE model are selected very close to physical realizations, this model is well suited to validate the considerably more abstract model from subsection 4.1 in the following section.

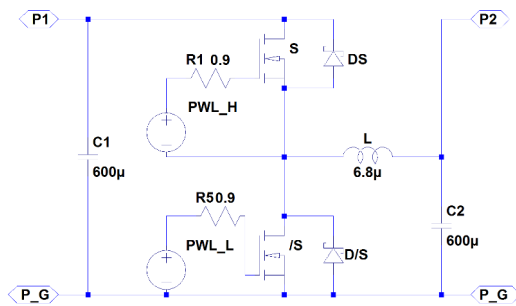


Figure 3. SPICE model of the proposed synchronous half-bridge converter

5 Validation of the proposed model

For parametrization of the models a synchronous half-bridge converter for a typical 48V / 12V conversion has been designed. The converter consists of the MOSFET *IPP052N08N5* (Infineon, 2014), driven by *UCC27210* (Texas Instruments, 2014) the inductance *SER2918H-682* (Coilcraft, 2014) and 60 µF/3 mΩ-capacitors, as well as a Schottky diode *MBR20100CT* (On Semiconductor, 2015) for the SPICE simulation. The values that are required for the parametrization of the model can be taken from the data sheets and are listed in Table 1. The SPICE simulations were carried out with the software *LTSPICE IV* by *Linear Technology*. The simulation environment for the Modelica model is *Dymola* by *Dassault Systèmes*. During the simulations for the validation, the significant speed advantage became apparent. For the computation of an inter-

val of 100 ms the SPICE model takes about 10 min, whereas the Modelica model takes less than 100 ms using an interval length of 10 µs.

Table 1. Model parameters

Name	Value	Description
f_s	200 kHz	Switching frequency
R_{DS}	5.2 mΩ	On-Resistance of the switch
Q_g	42 nC	Total gate charge
$R_{g,total}$	2 Ω	Sum of all gate resistances
V_{drive}	12 V	Gate-drive voltage
L	6.8 µH	Inductance of the inductor
R_L	2.6 mΩ	Resistance of the inductor
C_1, C_2	60 µF	Capacity of C_1 and C_2
R_{C1}, R_{C2}	3 mΩ	ESR of C_1 and C_2

5.1 Dynamic behavior

In this subsection the dynamic behavior of the proposed model shall be compared with the SPICE model on the basis of step responses on the load current. Therefore, the duty cycle is kept constant and the load resistance is changed to a different load scenario.

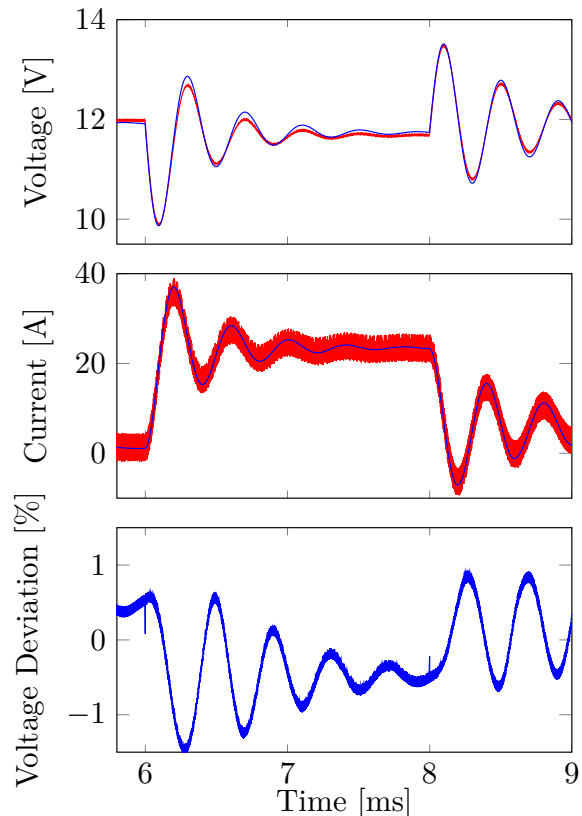


Figure 4. Output voltage and inductor current step response of the Modelica (blue) and the SPICE model (red) to a step in the load resistance during buck mode and the relative voltage deviation between the Modelica to the SPICE model

The first scenario examines the buck operation. The input v_1 is fed by a constant voltage source of 48 V with an internal resistance of 10 m Ω , the duty cycle is constant at 25%. In steady state, the load resistance is reduced to 0.5 Ω and after another 2 ms increased to 2 Ω . The result of the simulation is shown in Figure 4. The output voltage and the inductor current are illustrated in the upper two graphs, red for the results of the SPICE simulation, blue for the Modelica model. The lower graph shows the relative error of the output voltage that has a maximum relative error of 1.5% (absolute 200 mV).

The second scenario examines the boost operation. Now the input is v_2 and is fed by a constant voltage source of 12 V also with an internal resistance of 10 m Ω , the duty is held constant at 25%. In steady state, the load resistance is reduced to 8 Ω and after another 2 ms increased to 24 Ω . The result of the simulation is shown in Figure 5. The output voltage and the inductor current are illustrated in the upper two graphs, red for the results of the SPICE simulation, blue for the Modelica model. The lower graph shows the relative error of the output voltage that has a maximum relative error of 1.5% (absolute 720 mV).

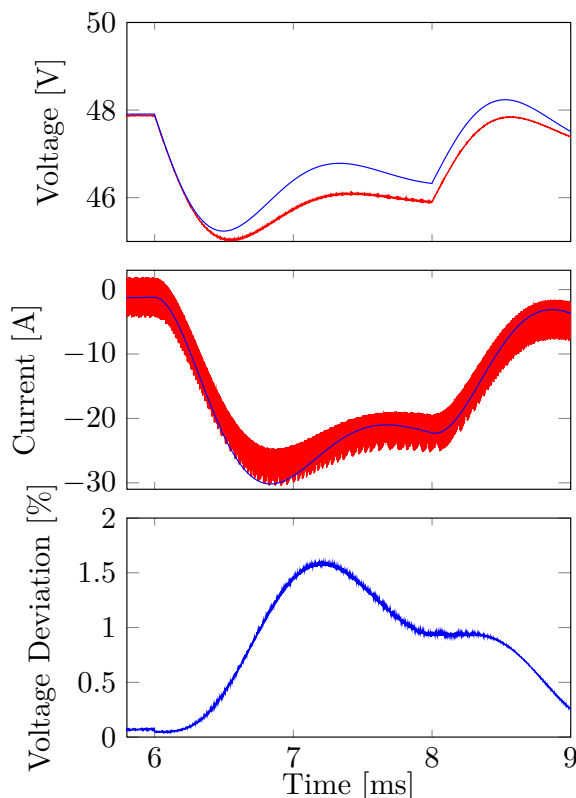


Figure 5. Output voltage and inductor current step response of the Modelica (blue) and the SPICE model (red) to a step in the load resistance during boost mode and the relative voltage deviation between the Modelica to the SPICE model

5.2 Efficiency and losses

In this section, the resulting efficiency of the proposed model shall be compared with the efficiency of the SPICE model. Figure 6 shows the efficiency map of the Modelica model in buck mode as a function of the output current and the duty cycle. The input is fed by a constant voltage source of 48 V.

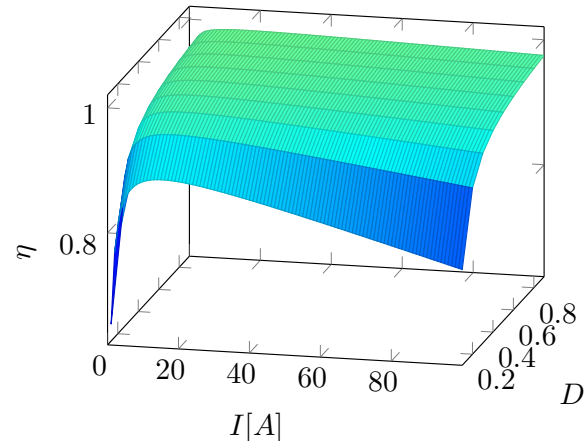


Figure 6. Efficiency map of the proposed Modelica model in buck mode. The input voltage is fixed at 48 V.

The result is the typical efficiency map of a DC/DC converter, having a bad efficiency at low currents, an maximum in the lower quarter of the output current and a subsequent slight lowering of efficiency at higher loads, as there ohmic losses are the dominating effect. Figure 7 shows the absolute deviation

$$\Delta\eta = \eta_{Dymola} - \eta_{SPICE} \quad (16)$$

between the SPICE and the Modelica simulation. It can be seen that in the range of high duty cycles and moderate to high output currents, the absolute deviation is lower than two percentage points. At very low duty cycles there is a trend towards a higher deviation as well as in the area of very low output currents. The negative deviation in the low-current range may be caused by the fact that charging effects of the MOSFET's and diode's parasitic capacitors are not yet implemented in the Modelica model. Reasons for the positive deviation in the range of higher currents are the not yet implemented dead-time losses as well as the effects of ringing in the switching node.

6 Conclusion and Outlook

In the previous considerations, a model of a synchronous half-bridge converter was presented. The model represents losses and the dynamic behavior given by inductors and capacitors of the converter,

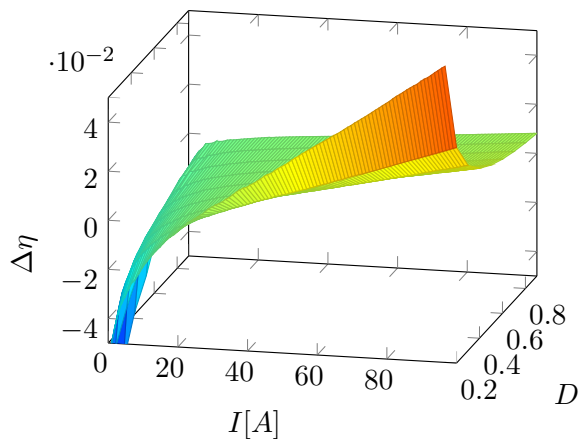


Figure 7. The absolute efficiency deviation $\Delta\eta$ of the SPICE model against the proposed Modelica implementation

without the need of simulating every switching process of the power semiconductors. This approach reduces the calculation effort of the model by several magnitudes. The model can be parametrized easily by using datasheet parameters of the components used in the design. To validate the Modelica model, an equivalent model in SPICE is used which is much closer to physical reality as it takes the switching behavior of the converter into account and uses more detailed semiconductor models provided by the manufacturers. The deviation between both models concerning dynamic behavior during load steps was investigated. It was shown that there is a maximum relative error of 1.5% in the output voltage of the model in reference to the SPICE model. Furthermore, also the energetic behavior was investigated. It was shown here that the proposed model provides good results for specific operating areas. Still there are some effects which should be considered in future examinations in order to improve the accuracy of the model. These are for example losses because of the reverse recovery effect of the diodes or losses due to ripple currents in the capacitors.

References

- Coilcraft. *Datasheet - Shielded Power Inductors - SER2900*, 2014.
- Robert W. Erickson and Dragan Maksimović. *Fundamentals of power electronics*. Kluwer Academic, Norwell, Mass., 2nd ed edition, 2001. ISBN 0-306-48048-4.
- Harald Giuliani, Claus-J. Fenz, Anton Haumer, and Hansjörg Kapeller. Simulation and validation of power losses in the buck-converter model included in the smartelectricdrives library. In *The 8th International Modelica Conference, Technical University, Dresden, Germany*, Linköping Electronic Conference Proceedings, pages 369–374. Linköping University Electronic Press, 2011.
- J. S. Glaser, F. E. Cellier, and A. F. Witulski. Object-oriented power system modeling using the dymola modeling language. In *PESC '95 - Power Electronics Specialist Conference*, pages 837–843, 1995. doi:10.1109/PESC.1995.474914.
- Johannes Gragger. Computation time efficient models of dc-to-dc converters for multi-domain simulations. In Matthias Schmidt, editor, *Advances in Computer Science and Engineering*. InTech, 2011. ISBN 978-953-307-173-2. doi:10.5772/15813.
- Infineon. *Datasheet IPP052N08N5*, 2014.
- Marian Kazimierczuk. *Pulse-width modulated DC-DC power converters*. Wiley, Chichester, U.K., 2008. ISBN 978-0-470-77301-7.
- Tom P. Kohler, Thomas Wagner, Andreas Thanheiser, Christiane Bertram, Dominik Buecherl, Hans-Georg Herzog, and Joachim Froeschl. Experimental investigation on voltage stability in vehicle power nets for power distribution management. In *2010 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–6, 2010. doi:10.1109/VPPC.2010.5729168.
- Kristin Majetta, Sandra Böhme, Christoph Clauß, and Peter Schneider. Msl electrical spice3 - status and further development. In *The 8th International Modelica Conference, Technical University, Dresden, Germany*, Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2011.
- On Semiconductor. *Datasheet - MBR20100CT - Switch-mode Power Rectifiers*, 2015.
- Florian Ruf, Alexander Neiss, Andreas Barthels, Tom P. Kohler, Hans-Ulrich Michel, Joachim Froeschl, and Hans-Georg Herzog. Design optimization of a 14 v automotive power net using a parallelized direct algorithm in a physical simulation. In *13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, 2012.
- Florian Ruf, Markus M. Schill, Andreas Barthels, Tom P. Kohler, Hans-Ulrich Michel, Joachim Froeschl, and Hans-Georg Herzog. Topology and design optimization of a 14 v automotive power net using a modified discrete pso in a physical simulation. In *2013 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–7, 2013. doi:10.1109/VPPC.2013.6671740.
- Texas Instruments. *UCC2721x 120-V Boot, 4-A Peak, High Frequency High-Side and Low-Side Driver (Rev. F)*, 2014.
- Vishay. *DC-to-DC Design Guide*. Vishay Siliconix, 2002.

Modeling and Simulation of Liquid Propellant Rocket Engine Transient Performance Using Modelica

Liu Wei¹ Chen Liping¹ Xie Gang¹ Ding Ji² Zhang Haiming² Yang Hao²

¹School of Mechanical Science & Engineering, Huazhong University of Sci. & Tech., China,
liuwei20@foxmail.com, {chenlp, xieg}@tongyuan.cc

²Suzhou Tongyuan Software & Control Technology Co., Suzhou, China,
{dingj, zhanghm, yangh}@tongyuan.cc

Abstract

This paper presents a liquid propellant rocket engine (LPRE) model library in Modelica language, which contains component models such as pipes, valves, tanks, turbo-pumps, combustion chambers, nozzles, injectors, gas generators, etc. These component models can be applied to establish a variety of liquid rocket engine systems with the capability of predicting engine transient performance during the startup, shutdown and regulation processes. Typical gas-pressurized liquid propellant engine system and turbo-pump liquid propellant engine system are modeled in the paper. Some simulations and analyses are performed to validate the models qualitatively. All the modeling and simulations are implemented in MWorks (Zhou, 2006), which is a modeling and simulation platform that fully supports Modelica.

Keywords: liquid propellant engine, thermo-fluid, startup and shutdown transient

1 Introduction

Liquid propellant rocket engines are widely used and play a very important role in aerospace. The function of a LPRE is to generate thrust through chemical reactions, which usually release thermal energy from the chemical energy of the propellants. The pressure generated from the thermal energy imparts a momentum to the reaction products. Then a momentum in the opposite direction is imparted to the rocket and propels a vehicle in space. A LPRE system usually consists of thrust chamber assembly, propellant feed system, turbine-drive system (for turbo-pump LPRE), and propellant control system, etc. A liquid propellant rocket engine is very complex and difficult to design and analyze because of many coupled subsystems and their extreme working conditions. Physical experiments under various conditions are also expensive. Hence it's critical to utilize models to facilitate the design and analysis process of LPRE. The control equations of LPRE dynamic motion are implicit and nonlinear differential algebraic equations. The structure and components of one LPRE often

differ from those of another. Therefore, it is a big challenge for engineers to build LPRE models that are of high generality and reusability, flexibility. Most of the existing models for LPRE (Karimi, 2003; Ruth, 1990; Matteo, 2012; Tabrizi, 2013; Karimi, 2006) lack generality, reusability or flexibility. A usual completed system model can only be applied to some specific LPRE and cannot be modified directly to be applied to others. Besides, the modeling process is difficult and time consuming, because engineers have to consider all of the numerical problems in equations solving procedure.

Modelica (Fritzson, 2010) is an object-oriented equation-based modeling language, which is capable of multi-domain modeling and has a strong software component model with structure for creating and connecting components. Modelica allows engineers to use mathematics to define system behaviors naturally and have powerful structuring capability to deal with complex interconnected systems. Engineers often do not need to consider numerical solving problem, thanks to the Modelica developing environment. These properties make Modelica suitable for the modeling and simulation problems of large scale and complex LPRE system. This paper focuses on modeling general library and efficient simulation of LPRE using Modelica.

2 Implementation of component models

We try to make best use of capabilities of Modelica when developing the liquid propellant rocket engine library. Firstly, we determine the objective of the library. A system model is expected to predict the flow rate, pressure and temperature of the components in a LPRE system during the whole running time. The dynamics of LPRE mainly consists of fluid dynamics, heat transfer, thermal dynamics and combustion, all of which should be taken into consideration.

According to the natural border in the LPRE system and object orientation, we divide a system into interacting components. Object orientation is viewed as a structuring tool to handle the topological structure description of a LPRE system. In order to decrease

complexity, we assume that the decomposed components are independent physical functional objects, including pipes, valves, tanks, turbo-pumps, combustion chambers, nozzles, injectors, gas generators and bottles. While subject decomposition (Jensen, 2003) is also performed to obtain base models, which collect common properties of a class of models and are physical phenomenon units. The main base models in LPRE library consist of control volume, flow model, ideal gas property, heat convection, heat conduction and combustion model. These base models cannot be simulated directly, and are inherited and aggregated by more than one component model for reusing.

The components models exchange information through connectors. The connectors ensure that components are independent of each other and work under a set of boundary conditions provided by connectors. The connectors of component models should present the properties of interactions between these components in a real physical LPRE system. Thus it would be easy and natural to connect components. The LPRE library contains four connectors for fluid flow, heat flow, 2D rotation and 2D translation respectively. In order to make the LPRE library consistent with Modelica Standard Library (MSL) and increase versatility, the four kinds of the connectors in the LPRE library are the same with those in MSL. Using base models, connectors and mathematical models of physical components, detailed component models are developed and implemented.

3 Description of mathematical models

In this chapter, the mathematical models of some most important components are presented, including combustion chamber, nozzle, pipe, valve, pump and turbine.

3.1 Combustion chamber

The thrust chamber is a key subsystem of a LPRE. The combustion chamber is a part of the thrust chamber where the chemical reaction of the propellant takes place to generate hot gas products. It is assumed that liquid propellants react and change to hot gas after a constant delay time and gas flow in combustion chamber is adiabatic. The control equations of a bipropellant combustion chamber are described as follows.

$$\frac{dm_{ox}}{dt} = \dot{m}_{oxi} - \frac{m_{ox}}{\tau} \quad (1)$$

$$\frac{dm_{fu}}{dt} = \dot{m}_{fui} - \frac{m_{fu}}{\tau} \quad (2)$$

$$\frac{dm_g}{dt} = \frac{m_{ox} + m_{fu}}{\tau} - \dot{m}_{go} \quad (3)$$

$$\frac{dK_m}{dt} = \frac{1 + K_m}{m_g} \left(\frac{m_{ox} - m_{fu}}{\tau} \right) \quad (4)$$

$$\frac{dp}{dt} = \frac{RT}{V} \frac{dm_g}{dt} + \frac{p}{RT} \frac{d(RT)}{dt} + \frac{p}{V} \frac{dV}{dt} \quad (5)$$

$$\tau_g \frac{dRT}{dt} = RT_i(p, K_m) - RT \quad (6)$$

Here m_{ox} is the oxidizer mass, m_{fu} is the fuel mass, \dot{m}_{oxi} and \dot{m}_{fui} are the inlet mass flow rates of oxidizer and fuel, \dot{m}_{go} is the outlet mass flow rate of gas products, K_m is the propellants mixture ratio, p is the pressure inside chamber, RT is the product of gas constant and temperature of gas staying in combustion chamber, $RT_i(p, K_m)$ is the product of gas constant and temperature of gas products defined as a function of pressure and mixture ratio, τ_g is the stay time of gas in chamber, V is the volume of chamber. The control equations of a gas generator resemble those of the combustion chamber.

3.2 Nozzle

The nozzle is also a part of a thrust chamber, whose function is to accelerate gases and create high exhaust velocity. It is assumed that the gas flow through the nozzle is an isentropic expansion. Nozzle expansion ratio is defined as follows:

$$\varepsilon = \frac{A_e}{A_t} = \frac{\left(\frac{2}{\gamma + 1} \right)^{\frac{1}{\gamma-1}} \frac{p_c}{p_e}}{\sqrt{\frac{\gamma + 1}{\gamma - 1} \left[1 - \left(\frac{p_e}{p_c} \right)^{\frac{\gamma}{\gamma-1}} \right]}} \quad (7)$$

Here, A_e and A_t are the flow areas at nozzle exit and throat, p_c and p_e are the pressure at chamber and nozzle exit, γ is the specific heat ratio.

Velocity at nozzle exit is given by:

$$v_e = \sqrt{\frac{2g\gamma}{\gamma - 1} RT_c \left[1 - \left(\frac{p_e}{p_c} \right)^{\frac{\gamma-1}{\gamma}} \right]} \quad (8)$$

Here, R is the gas constant, T_c is gas temperature in combustion chamber.

The mass flow through a nozzle is given by:

$$\dot{m} = \frac{A_t p_c \gamma}{\sqrt{\gamma RT_c}} \sqrt{\left(\frac{2}{\gamma + 1} \right)^{\frac{\gamma+1}{\gamma-1}}} \quad (9)$$

The thrust force is defined as follows:

$$F = \dot{m} v_e + A_e (p_e - p_a) \quad (10)$$

3.3 Pipe

Pipes are interconnect components that carry fluid to the intended components. Pressure drop between inlet

and outlet of a pipe is determined by the following equation.

$$\Delta p = \lambda \frac{l}{d} \rho \frac{v^2}{2} + \xi \rho \frac{v^2}{2} \quad (11)$$

Here λ is the friction coefficient, ξ is the coefficient of the local head loss, l and d are length and diameter of the pipe, ρ and v are density and velocity of fluid in the pipe.

3.4 Valve

Valves control fluid flows. Every LPRE uses some of them. Liquid valves are governed by the following familiar equation, where the flow rate is the function of pressure drop and flow area:

$$\dot{m} = C_d A \sqrt{2\rho\Delta p} \quad (12)$$

Here C_d is the flow rate factor.

For a gas valve, when $\frac{p_o}{p_i} \geq \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}}$, we have

$$\dot{m} = C_d A \sqrt{\frac{2\gamma p_i \rho_i}{\gamma-1} \left[\left(\frac{p_o}{p_i}\right)^{\frac{2}{\gamma}} - \left(\frac{p_o}{p_i}\right)^{\frac{\gamma+1}{\gamma}} \right]} \quad (13)$$

When $\frac{p_o}{p_i} < \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}}$, then

$$\dot{m} = C_d A \sqrt{\frac{2\gamma p_i \rho_i}{\gamma+1} \left(\frac{2}{\gamma+1}\right)^{\frac{2}{\gamma-1}}} \quad (14)$$

3.5 Pump

A pump pressurizes propellants and deliver them to extended components in a turbo-pump propellant feed system. Centrifugal pump is the most widely used pump type. Performance maps for head and power are used in the pump model. The head h is evaluated by following equations:

$$h = \left(\frac{n}{n_{ref}}\right)^2 h_q \left(q \frac{n_{ref}}{n}\right) \quad (15)$$

$$P = \left(\frac{n}{n_{ref}}\right)^3 \frac{\rho}{\rho_{ref}} P_q \left(q \frac{n_{ref}}{n}\right) \quad (16)$$

$$P = \omega \tau \quad (17)$$

$$\eta_p = \frac{\Delta p q}{P} \quad (18)$$

Here, n is the rotational speed, q is the volume flow rate, P is the power consumption, ω is the angular velocity, τ is the torque, η_p is the pump efficiency, $h_q(\cdot)$ and $P_q(\cdot)$ are functions obtained from the performance map, ref is the reference value.

3.6 Turbine

A turbine gets energy from the expansion of high temperature and high pressure gas, and provides power to the pump. The expansion is assumed to be isentropic. The control equations of a turbine is presented as follows:

$$W = \frac{\gamma}{\gamma-1} RT_i \left[1 - \left(\frac{p_o}{p_i}\right)^{\frac{\gamma-1}{\gamma}} \right] \quad (19)$$

$$\eta_t = \eta_t \left(\frac{u}{C}\right) = \frac{\omega \tau}{W \dot{m}} \quad (20)$$

Here, W is the power of gas expansion, $\eta_t \left(\frac{u}{C}\right)$ is the turbine efficiency defined as a function of the velocity ratio obtained from the performance map. The mass flow rate in a turbine is evaluated by the familiar equation in the gas valve model.

4 Simulation and analysis of system

Liquid propellant rocket engines are classified into two major types according to their propellant feed system, namely gas-pressurized liquid propellant rocket engine and turbo-pump liquid propellant rocket engine. Typically, engines with small propellant quantities have a gas-pressurized propellant feed system, and large engines required weight considerations choose a turbo-pump propellant feed system. The startup and shutdown phases of a LPRE are very complex. The engine components are working under extreme operating conditions, and about half of the engine failures occur during the startup and shutdown. Thus the prediction of the transient characteristics of a LPRE is important and necessary to engine safety and reliability. In the next sections, we model two typical kinds of the LPRE, and perform simulations to obtain the transient characteristics of the LPRE.

4.1 Gas-pressurized liquid propellant rocket engine

A gas-pressurized LPRE consists of gas bottles, propellant tanks, pipes, valves, thrust chamber heads, injectors, combustion chambers, nozzles and igniters. According to the typical physical structure of gas-pressurized LPRE, a system model is quickly built by using component models in the LPRE model library. Figure 1 depicts the diagram view of the gas-pressurized LPRE system model. Figure 2 shows the combustion chamber pressure. Two thrust chambers start up and shut down at different times. There is a pressure pulse during the start-up, because the oxidant and the fuel flow into the combustion chamber asynchronously. Thus it is important to control the difference between the times when two propellants flow into combustion chamber initially, in order to

decrease the maximum of the pressure pulse. When the shut-down signal is given, the propellant control valves are closed. Pressure drops immediately, because the combustion lacks fuel and oxidant. The result also implies that the operations of one thrust chamber directly influence the steady pressure of another.

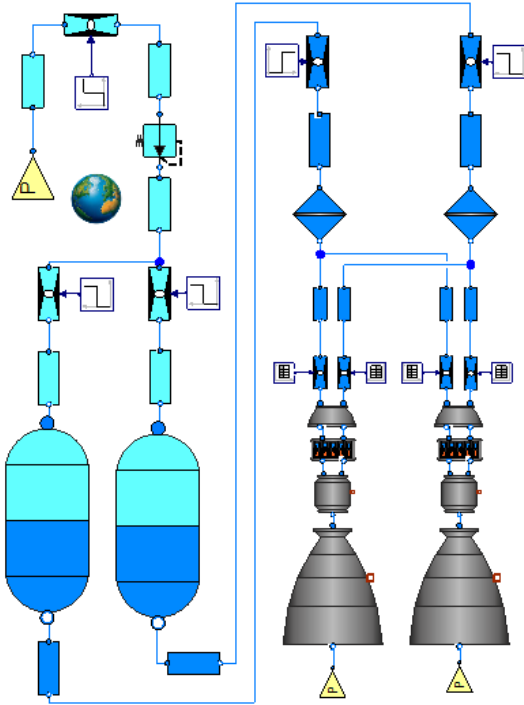


Figure 1. A gas-pressurized LPRE system Model

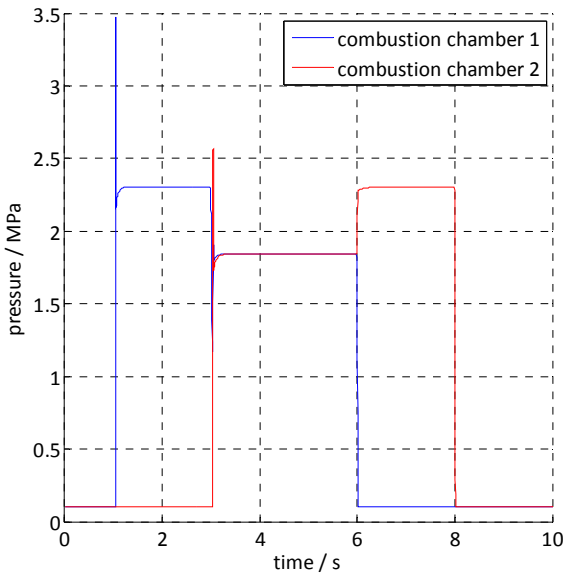


Figure 2. Pressure in the combustion chambers

4.2 Turbo-pump liquid propellant rocket engine

In contrast to the gas-pressurized LPRE, the turbo-pump LPRE has Turbo-pumps and a gas generator, but has no gas bottles. Figure 3 depicts the diagram view of a turbo-pump LPRE system model.

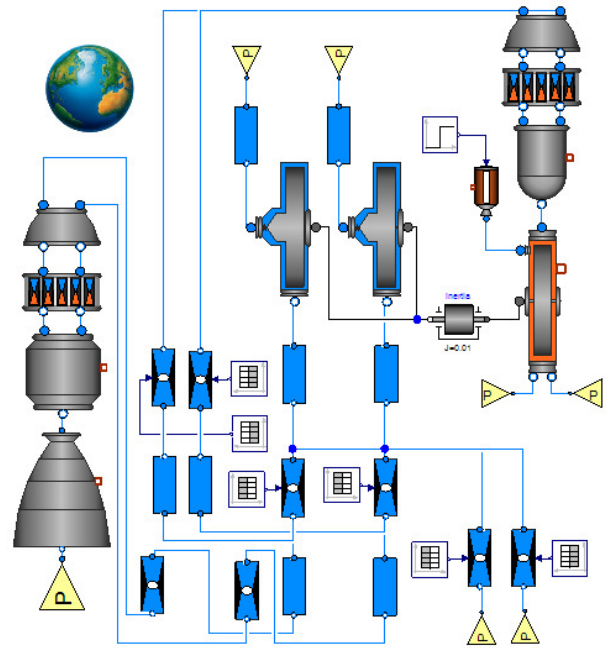


Figure 3. A turbo-pump LPRE system Model

Figures 4, 5 and 6 show the turbine shaft speed, pump outlet pressure, combustion chamber pressure and gas generator pressure during the start-up. Firstly, the igniter drives the turbine to run and the turbine rotational speed rises very quickly. The pump outlet pressure, as the function of the turbine rotational speed, also increases. The pumps then deliver propellants to the combustion chamber and gas generator. After the gas generator is ignited, it drives the turbo-pump in turn. Due to couple relations between gas generator and turbo-pump, rotational speed and pressure exceed the nominal ones, and then decrease and stabilize to the steady states.

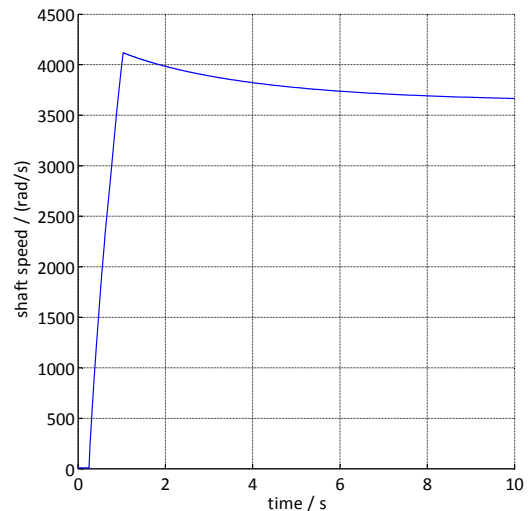


Figure 4. Rotational speed of turbine

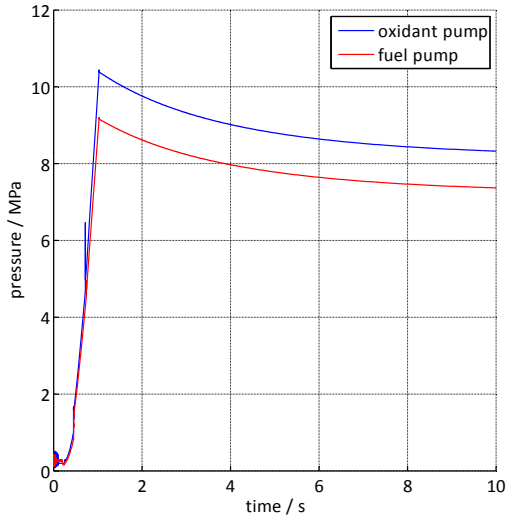


Figure 5. Pump outlet pressure

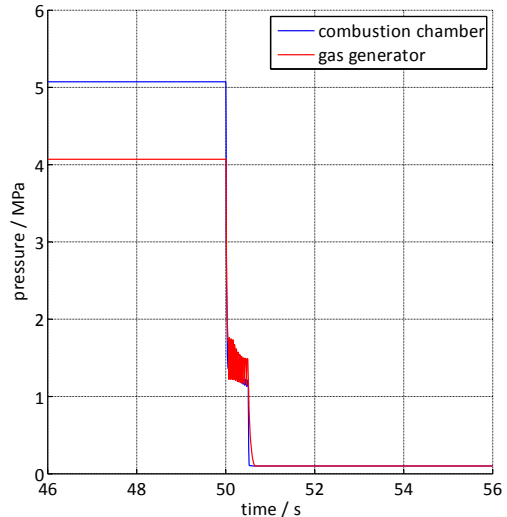


Figure 7. Pressure in combustion chamber and gas generator

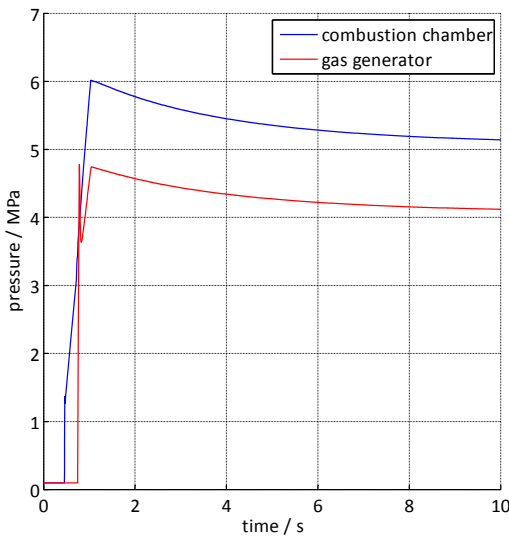


Figure 6. Pressure in combustion chamber and gas generator

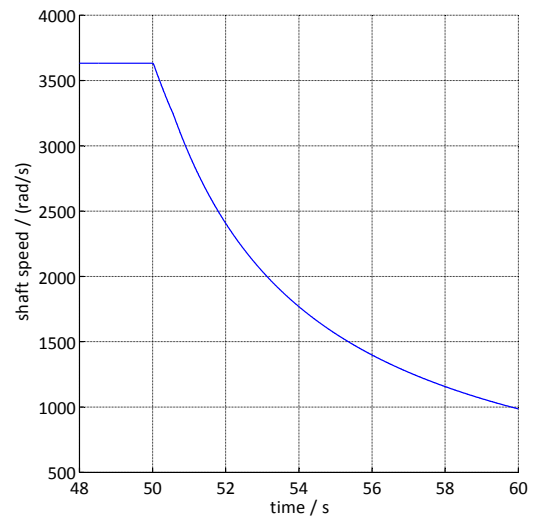


Figure 8. Rotational speed of turbine

Some interesting features of the shut-down process are depicted in Figures 7, 8 and 9. When the shut-down signal is given, the propellant control valves are closed. The gas generator lacks of propellants very soon, so the pressure in it begins to drop quickly. The rotational speed of turbine shaft decreases slowly, because the power delivered by gas generator to drive turbine get smaller and there exists resistance. There are residual propellants in the pipelines, thus pressure oscillations in the combustion chamber and gas generator are observed.

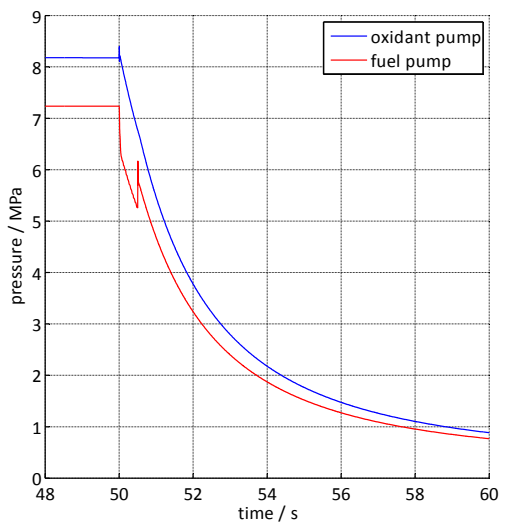


Figure 9. Pump outlet pressure

5 Conclusions

We have established a component model library for liquid propellant rocket engine that can be used to build LPRE system models efficiently and simulate engine transient performance. In this paper, we give the control equations of some most interesting components of LPRE. The general method for applying the characteristics of Modelica, especially object-orientation and connection mechanism, to the modeling procedure of LPRE is presented. Gas pressurized LPRE and turbo-pump LPRE system models are build using component models from the established LPRE library. The transients during engine start-up and shut-down are simulated and analyzed. Due to extreme working conditions and uncertainty, the start-up and shut-down processes are very complex. Our LPRE library provides an efficient tool to study the transient properties. In the future, we will validate the system model with existing experimental results and improve accuracy.

Acknowledgements

The paper is supported by the Key Project of National High Technology Research and Development Program (No. 2013AA041301).

References

- Fanli Zhou, Liping Chen, Yizhong Wu and et al. MWorks: a modern IDE for modeling and simulation of multi-domain physical systems based on Modelica. Proceedings of the 5th International Modelica Conference, Vol. 2: 725-731, 2006.
- Peter Fritzson. Principles of object-oriented modeling and simulation with Modelica 2.1. John Wiley & Sons, 2010.
- H. Karimi, A. Nassirharand, and M. Behesht. Dynamic and nonlinear simulation of liquid-propellant engines. Journal of propulsion and power, 19(5): 938-944, 2003.
- E. K. Ruth and R. L. Ahn. Advanced Liquid Rocket Engine Transient Model. 1990. AIAA-90-2299.
- Francesco Di Matteo, Marco De Rosa, and Marcello Onofri. Transient Simulation of the RL-10A-3-3A Rocket Engine. Space Propulsion Conference. 2012.
- Mahyar Naderi Tabrizi, Seyed Ali Reza Jalali Chime, and Hassan Karimi. Modeling and Simulation of Open Cycle Liquid Propellant Engines. Journal of Science and Engineering, 1(1): 17-34, 2013.
- H. Karimi, and A. Nassirharand. Application of a Simulation Algorithm for Dynamic analysis of a Liquid Propellant Engine. Journal of Aerospace Science and Technology, 3(1): 23-30, 2006.
- Jakob Munch Jensen. Dynamic Modeling of ThermoFluid Systems. Diss. Ph. D. thesis, Technical University of Denmark, 2003.

Model Based Specifications in Aircraft Systems Design

Martin R. Kuhn¹ Martin Otter¹ Tim Giese²

¹Institute of System Dynamics and Control, German Aerospace Center (DLR e.V.), Germany,
{martin.kuhn,martin.otter}@dlr.de

²Airbus operations GmbH, Germany, tim.giese@airbus.com

Abstract

This application paper describes the concept and needs on model based specifications in order to specify the basic behavior of aircraft systems and methods to check the requirements. It is demonstrated how it can be implemented by recent Modelica based libraries, especially with the new Modelica_Requirements library. Two new FFT-based requirement blocks are proposed to allow full coverage of the specification.

Keywords: executable specification, requirements, aircraft system design, FFT-based requirements.

1 Introduction

Executable specifications are computer algorithms written in an appropriate specification language with the purpose of demonstrating and verifying the compliance of the input-output behaviour of the model subject to the model specifications. In aircraft design, executable specifications demonstrating and verifying the behaviour of models can be seen as an important tool to make the co-work between airframers and suppliers quicker and more efficient, as they allow frequent testing and early validation of subsystems and systems interaction.

Similarly, requirement modelling allows the specification and testing of demands on signals which are generated by a system or the model of a system. Together, executable specifications and requirement models enable a well-defined specification of a system. Both methods allow testing against the hardware or software implementation. They strongly benefit from methods for monitoring and cross-checking.

While the traditional aircraft design process is based on document based specifications only, a model supported design process based on executable specifications and requirement models is thought to improve the process in terms of quality and time (Becker and Giese, 2011). In contrast to the traditional, more software oriented usage of executable specifications, here they were used in a more general way also for specification of physical models and behavior. In the publication the concept was evaluated with MathWorks based tools, but specification models may include physical models built with Modelica. In order to have a one-tool solution which allows better

coupling of the physical models to requirement blocks, alternatives to this approach with Modelica based methods were investigated in the “CleanSky, Systems for green operation” project (Cleansky, 2015). Associated tools were developed in parallel in the CleanSky subproject ModelSSA by Dassault Systèmes, supervised by DLR-SR and in the ITEA2 “MODRIO” project with several partners¹. This paper reviews the concept of executable specifications for aircraft systems² where the executable specifications are seen as a bigger package of specifications, test cases, demonstrators and monitoring functions. The implementation is solely based on Modelica.³

2 Review of model based design process

For aircraft systems design, the current design process is a document-based development. The behaviour of the system to be developed is defined by textual requirements, pseudo code, tables, block diagrams, logic diagrams and mathematical expressions. General demands applicable to several (sub-)systems are generalized in industrial standards, for example MIL-STD-704F (MIL704F, 2004) or airframer specific standards, for example the AirBus Directives (ABD). Document-based development has severe disadvantages: There is the danger of misunderstandings and misinterpretations of functional requirements since they are written in natural language which could result in incorrect system behaviour. Furthermore, the specified system behaviour cannot be simulated. Therefore, contradicting requirements can hardly be recognized before realization of the system. Also for multi-system functions and interfaces the validation is missing and therefore the mutual influences between systems may not be treated correctly in the early design cycles. This results in late detection of design errors when integrating the systems together. In addition, in case of requirements on signals and requirements on systems interacting with plants, the signal processing and plant test models may be implemented differently

¹ MODRIO: <https://itea3.org/project/modrio.html>

² In this paper aircraft systems (e.g. a drive) and components of a system (e.g. controller) are both called “system” to simplify notation.

³ Section 2-4 is based on the internal reports (Kuhn et. al., 2014; Becker et.al., 2013; Becker, 2014).

between the airframer and different suppliers. In any case there might be redundant work since the same monitoring functions need to be implemented by several suppliers or at the airframer for testing.

In contrast, by a Model Based Design Process (MBDP) the system to be developed is specified by models representing the functional and/or physical behavior. The models can be delivered with test environments and monitoring functions which are modeled by the airframer. To prevent confusions and double work, it is essential to tightly link the documentation and the model based specification. This can be achieved by automatic generation of the documentation from the model and its embedded requirements and optionally with linkage of models to requirements in databases. By this approach, misunderstandings and lack of information is avoided since the models provide a mathematically precise definition and allow interactive simulation and investigation. While the model should cover all aspects of the systems' functionality, there is no necessity to express all of it in a single model. A combination of methods as

- flow diagram notation,
- state transition notation,
- physical modelling, plus the afore-mentioned
- written requirements

can be used. This methodology was evaluated in (Becker and Giese, 2011). The efficiency of the model based approach was analyzed in (Becker et al., 2013). In (Becker, 2014) the model-based design process was tested qualitatively against the former development programs. It could be shown that the model based specification process results in reduced cost for development of control systems. Those are significant advantages from a project management perspective.

3 Elements of an executable specification model

In the following we will introduce representative requirements in the style of (Tunnat, 2011) and (MIL704F, 2004) for the Environmental Control System (ECS) and for the electrical system.

For a model-based design process, the requirements can be grouped into two different layers:

- The high level requirements.
- The functional requirements.

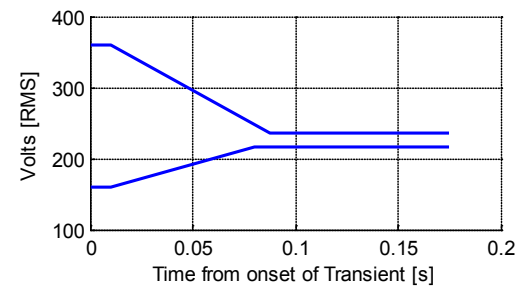
The high level requirements treat specifications on the exterior behavior of the system. The formulation is based on engineering knowledge and top level demands. The **high level requirements** include

- (1) Demands on the implementation and realization. Those requirements generally are not stated by use of models. A requirement can be stated as in R1.

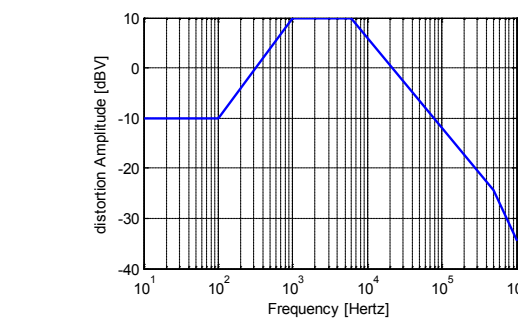
R1: The engine's total probability of failure must be smaller as $1e-9$.

- (2) Demands on the signal and state behavior. Such requirements are stated by requirement blocks that assess the specific requirements using observation variables from a physical model as part of the executable specification. Requirements may be specified based on industrial standards. A requirement can be stated as in R2 and R3.

R2: In normal operation mode, the envelope of the RMS value of the 400 Hz AC voltage after a voltage transient is given by the following figure and has to remain in the final limits.



R3: In normal operation mode, the distortion spectrum of the variable frequency AC voltage has to remain below the limits given by the following figure.



In contrast to the high level requirements, the **functional requirements** define the realization and logic of the system to be realized in an abstract but executable language. For example, R4 and R5:

R4: In case Ditching is not active, the OVBDV shall be in its PO position and the BUV shall be in its FO position, five seconds after Override has been activated

R5: For a two-position valve defining a valve flap the following functional behaviour shall apply:

If the system is in state "open" indicated by "full_open"=true, a commanding signal "closing" without indication "fault" shall result in state "close". In case of "fault" the system shall go into condition "open_fault" with indicator "stuck_open"=true. Reciprocal rule for state "close" with indicator "full_close", command opening and fault condition "close_fault" indicated by "stuck_close"

In these examples, the logic is functional as no details on the physical realization is given.

A fully model based design process relies on extensive modelling to express demands by functional modelling, supplied test environments, model of the physical system with its components, test it against the specification and check the result. The center in charge of the aircraft or system specification may cover all or a selection of the following tasks to express the model based specification for a component (or system) which shall be developed:

Table 1: Elements of model based specification process

1	Specification of the components functional and procedural behavior by models.
2	Simple physical model to demonstrate the desired behavior of the component and allow simulation with physical test environments.
3	Physical modelling of the testing environment.
4	Expression of requirements and modes of operation by requirement monitors.
5	Definition of interfaces to physical states, environmental states, logical states.
6	Mapping of requirements to the interfaces of the functional models.
7	Providing property monitors with built in signal operations for requirements which demand advanced processing of interface signals-
8	Providing indicators and automatic documentation of warnings and faults.
9	Make tools available for managing requirements and documentation

In the systems realization phase of the supplier and afterwards in the systems integration phase of the airframer there are additional demands for

- Systematic testing of system models.
- Clearance of requirements.

It is the task of the supplier to realize the system and harmonize the behavior of the executable specification and the developed system. For this, the model of the developed system can be embedded into the test model, being optimized and checked by the monitors.

4 Realization with tools of MathWorks

The aforementioned approach was evaluated by Airbus Germany at hand of a controller design of the ECS. The model of the controller could be best modelled by hierarchical state charts and stateless flow charts.

The modeling platform used several packages and tools of the MathWorks product family. The **physical system** of the ECS system architecture was modelled with Simulink. Alternatively, Modelica models can be imported in Simulink. For the hybrid **state space modeling** of component models, Stateflow was used (MathWorks, 2015b).

The **functional specifications** of the controller make use of Stateflow as well. The state diagrams give a detailed description on the systems behavior,

including start sequence, transitional conditions and entry conditions when changing to adjacent states.

For managing of requirements, no ready to use product was found which met the demands. Thus a special requirement manager was commissioned by Airbus (toolbox developed by Silver Atena⁴).

The requirement manager summarizes and documents the requirements, tracks the requirements changes and allows some coverage analysis on requirements with predefined test scenarios. It relies on additional special properties block which are inserted to the local functional model. For this part the "Verification and Validation" toolbox (MathWorks, 2015a), the Airbus requirement manager, and Simulink's "Report Generator" is used.

An example of a model based specification for an ECS controller is shown in Figure 1 formally defining Requirement 5. The pneumatic network is the physical plant which has to be stabilized by a controller to be developed. The pneumatic system acts as environment model and is realized by Simulink blocks. The preliminary model of the controller can be implemented in a very simple manner at this stage. The only aim is that the physical system can be simulated, even if the simulation results violate requirements. For example, in case a physical demonstrational model is needed, the controller could be implemented as a P controller while the supplier's realization may rely on a sophisticated model-based controller.

In addition to the preliminary controller - and more important - the functionality of the controller (R5) is defined by additional Stateflow diagrams. They are the result of a pre-design at the airframer. In the right part of Figure 1, requirements for the behavior in case of errors are defined. The system can be simulated and checked interactively by variation of the input states of the Stateflow system.

No special monitors for high level requirements were implemented.

After implementation of the functional executable specification, the formal verification of requirements can be realized with the "Design Verifier" block set from MathWorks. An example is shown in Figure 2 formally specifying requirement R4.

The blocks in the left calculate the requirement while the „statement“ block is linked to a requirement checker and monitoring system. The system supports documentation and formal verification of the requirements. Other special monitors for high level requirements can be implemented with Simulink blocks or Simulink S-functions.

⁴ <http://www.silver-atena.de>

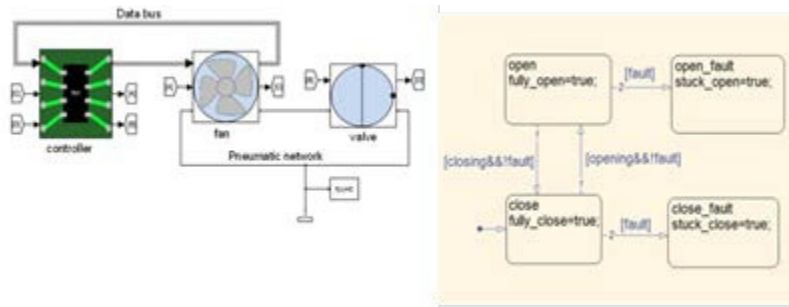


Figure 1: Model based specification at hand of an ECS example

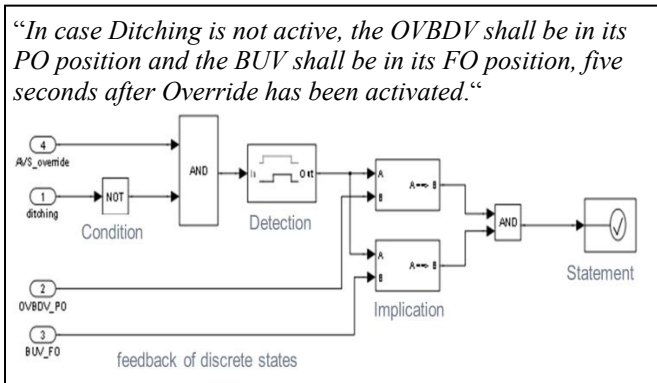


Figure 2: Formal specification of requirement R4 using the Design Verifier from MathWorks. Figure and text from (Tunnat, 2011).

5 Realization with Modelica

This section shows how to use Modelica for modelling and checking of requirements to provide the necessary functionality of Table 1. In general, the transition from the paper based design process to model based specification, executable specifications and automated testing is mostly a matter of the development philosophy rather than of the technical realization.

Functional requirements can be most conveniently specified in Modelica by synchronous state machines (Elmqvist et al., 2012). In Figure 3 the example of Figure 1 is shown, implemented with a Modelica synchronous state machine.

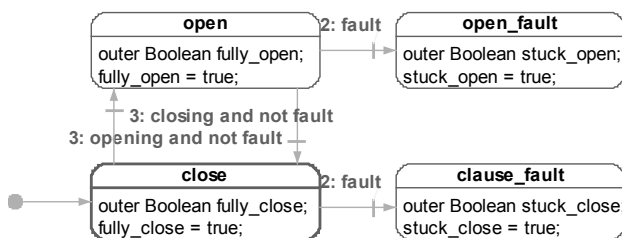


Figure 3: Modelica synchronous state machine of the example in Figure 1.

The realization and user friendliness is mostly equivalent to a Stateflow implementation in this case. However, the Modelica synchronous state machines have a more rigorous definition to avoid modelling errors. For example, there may be assignments to the

same variable in different state machine “states” (such as in state “close_fault” in Figure 3. These state machine states are “mutually exclusive” and at one sample instant the code of only one of these states is executed. Furthermore, in parallel state machines, exactly one assignment to the same variable at the same sample instant is allowed. In essence, Modelica and a Modelica tool only allow one single assignment to the same variable at one sample instant, in order to always have deterministic, well-defined behaviour. On the other hand, in Stateflow several assignments to the same variable are possible.

Alternatively, one may express the system in Modelica by **behavior trees** which follow a slightly different concept. Behaviour trees can be used for modelling of logical behavior and especially mission planing. Complex missions are built up using atomic tasks. Tasks can query conditions from system states or trigger actions, e.g. by sending commands to the communication bus. For a detailed description of the Modelica library used here, see (Klöckner, 2014).⁵ The main advantage of behavior trees for executable specifications is their standardized and intuitive structure to express alternative paths. Plans are very scalable and human-readable on all levels of the hierarchy. It is their benefit and drawback at the same time to be inherently memory- and loop-free. They thus execute the correct task immediately after a restart or online modification.

This is demonstrated in Figure 6 which is the Modelica behavior tree implementation of the ECS example of Figure 1. Depending on the Boolean input variables *fault* and *opening*, one of the four conditions *open*, *close_fault*, *open_fault* or *close* occurs. The logic is like this: Starting always from the top, a *selector* tries to execute one of the paths linked below, where the preference is from left to right. The “*sequence*” starts a sequence from left to right, in case the breaking condition (II-Symbol) is true. For example in case of “*not opening*”, the “*selector*” cannot take the left path “*sequence*” which is blocked by “*condition*”. Instead the right path to *selector2* is tried. “*sequence2*” ends in the action “*open_fault*” in case “*faultyO*” is un-blocked by *fault = true*, and otherwise in “*close*”.

⁵ Different types of “behavior trees” in a Modelica context have also been used in (Myers, 2010).

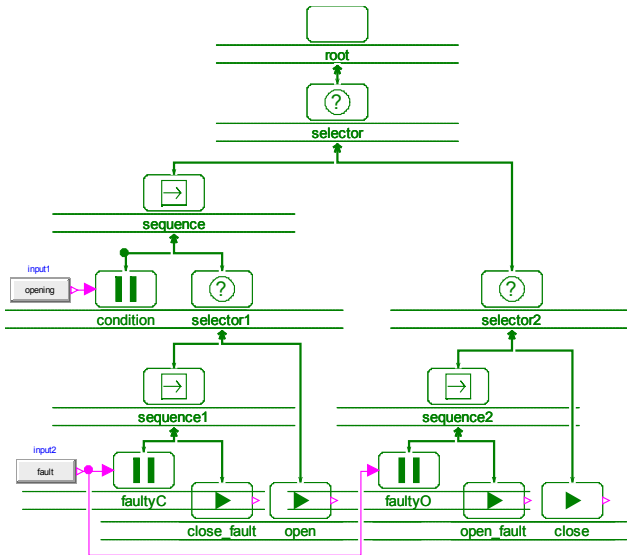


Figure 4: Modelica behavior tree of ECS example in Figure 1.

Another important demand of model based specification are physical demonstrator models of the system and modelling of test and environment models (Table 1, demand 2 and 3). Obviously, Modelica is very well suited for this part, due to the many available physical modeling libraries, that are much more intuitive to use than with only graphical input/output block diagrams.

The expression of high level requirements can be formulated in principle by any type of mathematical operation which results in an expression for requirement fulfilled/not fulfilled (or not yet evaluated). For example in (Kuhn, 2011) Modelica requirement models have been designed for band constraint signals or frequency domain constraints. The

textual output of the requirement checking was based on Dymola proprietary scripting and was missing systematic output and documentation concepts.

In parallel to JTI activities, the European ITEA projects EUROSYSLIB, OPENPROD, and their successor MODRIO also identified a strong need for requirements modelling. Their approach resulted in the new Modelica_Requirements library (Otter et al., 2015). One essential advantage of this library is that it uses two- and **three-valued logic** to specify requirements. It is then possible to distinguish whether a requirement is *satisfied*, *violated*, or *not tested* during a simulation. It could be demonstrated in the JTI project, that the requirements library fulfills many needs for formulation of executable specifications of the electrical and the ECS system. In particular, for the examples in this paper, the LogicalBlocks, the TimeLocators and the ChecksInSlidingWindow have been used.

Requirement R2 could be implemented with the Modelica_Requirements library with several BandDuration blocks. A more convenient approach is sketched in section 5.2 by using a newly designed and implemented “Funnel” block.

Frequency domain requirements, such as needed for Requirement R3, cannot be defined with the current Modelica_Requirements library. Therefore, new requirement blocks have been developed based on the Fast Fourier Transformation (FFT), see section 5.3.

An implementation of requirement R4 with the Modelica_Requirements library is shown in Figure 5. The requirement block “*requirement_AVs_override*” displays the textual version of the requirement in its icon and collects the status of all requirement blocks during one simulation run. By this example it is also o

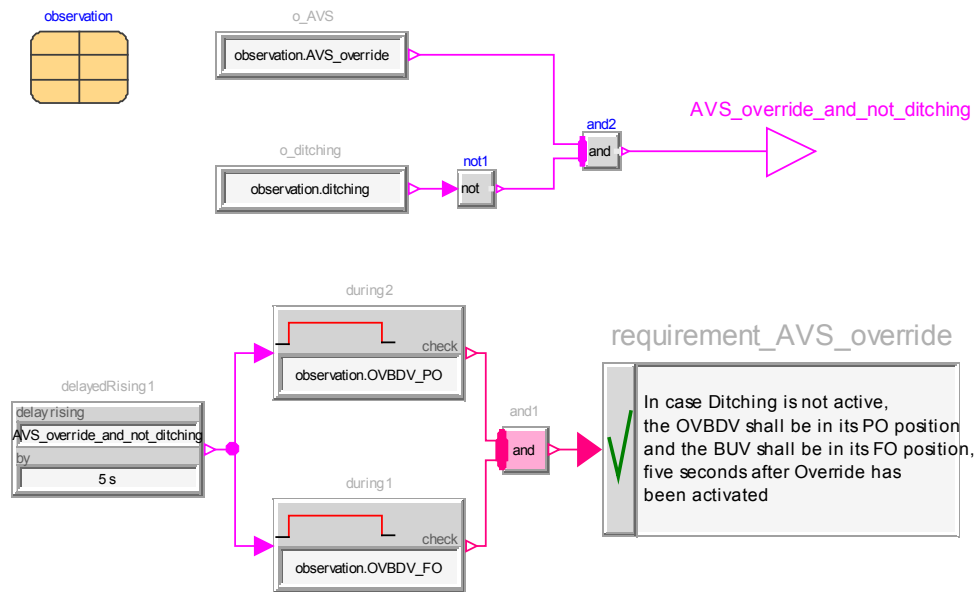


Figure 5: Formal specification of requirement R4 with the Modelica_Requirements library.

demonstrated how to bind requirements to the physical model: The general idea is to define observation variables in Modelica records, as needed from a physical system model (“observation” in Figure 5). Via newly developed Modelica language elements the actual values of the observation variables can be inquired conveniently from the physical system model (Elmqvist et al., 2015). Figure 6 shows the final status with the requirement model (lower left) and the system model (upper part). The system model may be the executable specification or the supplier’s model in the verification phase.

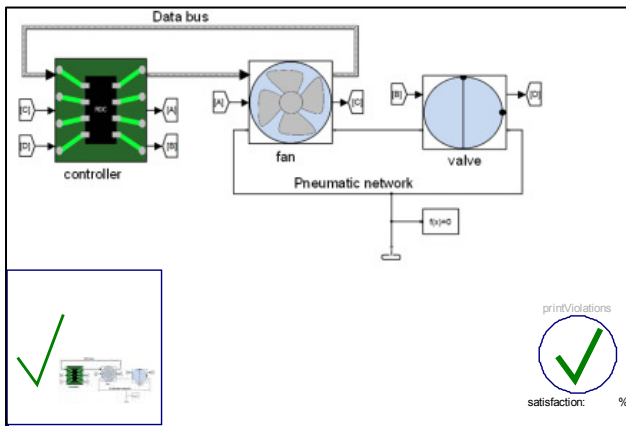


Figure 6: Binding and assessment of ECS requirements.

The requirement model is linked to the system model by the following instantiation of the requirement:

```
Requirements.AVSRequirements Req1(
  observationName="controller",
  observation= Bindings.AVSRequirementFromController(
    controller))
```

`Bindings.AVSRequirementFromController` is a function to map the variables from the controller to the requirement record. `observationName` defines the name of the target of the requirement. This is needed for automatic documentation.

At the end of the simulation, the following log is displayed:

```
--- 100 % of the requirements are satisfied ---
Requirements satisfied (1 of 1):
Controller(Req1.requirement_AVS_override):
In case Ditching is not active, the OVBDV shall
be in its PO position and the BUV shall be in
its FO position, five seconds after Override has
been activated
```

The current development stage allows to check in every simulation run whether the defined requirements are satisfied or violated (or are not tested).

The binding concept is flexible enough to bind requirements to all instances of a class using the experimental component iterators. For example in case of multiple controllers, an iteration would map a requirement to each controller.

For organization of functional expressions, there exists no requirement manager similar to the Simulink

solution for Modelica yet. The Simulink tool summarizes and documents the requirements, tracks the requirements changes and allows coverage analysis on requirements with predefined test scenarios. In the MODRIO project, further developments are planned in this direction.

5.1 Demonstration: Specification of hardware

As last example, the concept of a model based design process relying on a model based specification shall be demonstrated at hand of a realistic example of the design of a generator. Alternatively to written specifications, the airframer may deliver a model based specification. The test model is shown in Figure 7 on the right side (supply of linear resistive three phase load and nonlinear rectified load to investigate the harmonics in the AC line).

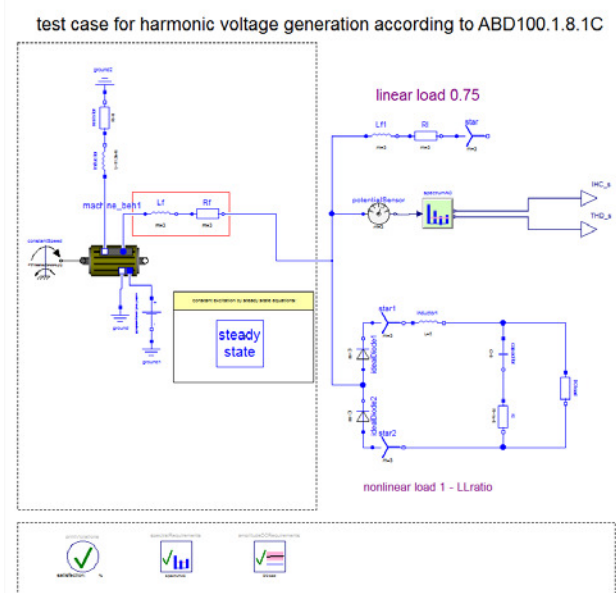


Figure 7: Demonstration of testing environment with requirement models and signal monitors.

The availability of test models allows easy and uniform implementation for all suppliers. Special operations on signals needed for the requirements checking might be also given as models. Here, the green block embeds an FFT based requirements blocks, an alternative realization (Kuhn, 2011) to the FFT block of section 5.3.

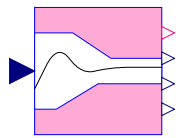
The requirements are stated by Modelica blocks. In this case two requirement blocks are associated to the model in the lower right corner which check the requirements for the operating area of the DC voltage and frequency content of the AC line voltage.

A primitive generator model might be supplied by the airframer. This is replaced by the supplier by a much more detailed model (dashed box in the left). By this test environment, the generator model can be tested and also optimized in relation to the requirements.

The model in the left lower corner triggers the summary log of the requirement blocks. The output together with the documentation of the test model and the system model (generator) are valuable parts of a proper industrial model delivery.

5.2 Transient Limits monitor

The “Funnel” block, displayed in the figure to the right, allows checking of transient time limits in funnel style. The upper and lower limits are defined via a table versus time. The initial start of the time varying limits is triggered by an initial overshoot of the limits. The initial limits are defined by the final band. This funnel type limit may be retriggered if one full period of the funnel style limitation has gone by. The output *y* indicates the satisfaction of the criterion. Further outputs are a scaled distance to the limits and the time varying upper and lower limits.



5.3 FFT-based frequency property monitor

Frequency based criteria are typical for industrial standards of electrical systems but are not yet supported in the Modelica_Requirements library. For example, MIL-STD-704F (MIL704F, 2004) defines a maximum distortion in the spectrum of the 270Volts DC system.

Based on the implementation and practical experience with the FFT monitoring block of (Kuhn, 2011), two FFT blocks were newly designed and implemented. An example of the user’s view of the new FFT block *WithinAbsoluteFFTdomain* is shown in Figure 8. An alternative block with limits for total harmonic distortion (THD) is shown in Figure 9.

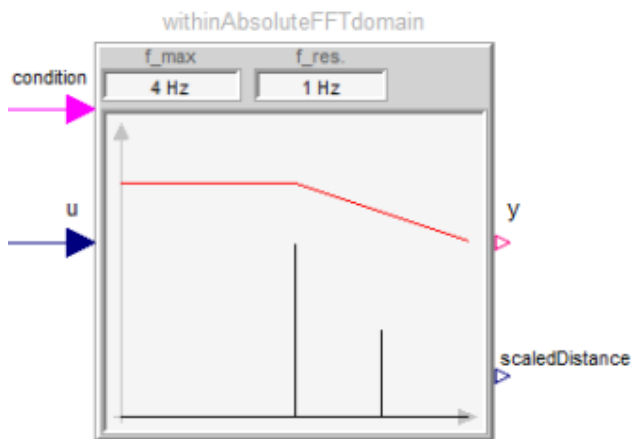


Figure 8: Example of *WithinAbsoluteFFTdomain* block for the inputs: $u = 2 + 3 \cdot \sin 2\pi f_1 t + 1.5 \cdot \sin 2\pi f_2 t$ ($f_1 = 2 \text{ Hz}, f_2 = 3 \text{ Hz}$) and **condition = true**.

The user interfaces were designed to allow parameterization with a minimum of information and display the amplitudes over the frequencies in the icon.

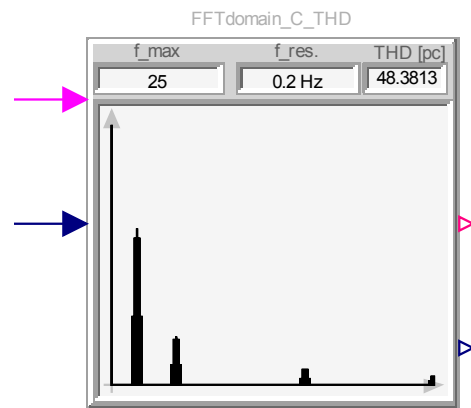


Figure 9: Example of *WithinAbsoluteFFTdomain_THD* block with: $u = 5 + 3 \cdot \sin 2\pi f_1 + 1.5 \cdot \text{pulse}(2\pi f_2)$ ($f_1 = 2 \text{ Hz}, f_2 = 5 \text{ Hz}$, “pulse” is the rectangular pulse function at frequency f_2) and **condition = true**.

In the icon of *WithinAbsoluteFFTdomain*, the two scalar parameters of this block are displayed, f_{max} – the maximum frequency of interest for the user, and f_{res} – the resolution of the frequency axis (so the increment of the frequency axis). Typically, the user is interested in a maximum frequency f_{max} that is an integer multiple of some base frequency (e.g. 50 Hz base for power distribution networks). The frequency resolution should be selected in such a way, that the spectral lines of particular interest are an integer multiple of the resolution (in order to get the most accurate result). In the example $f_{\text{max}} = 4 \text{ Hz}$ and $f_{\text{res}} = 1 \text{ Hz}$, so 5 frequency values are shown in the icon (0, 1, 2, 3, 4 Hz). For numerical reasons, in practice the resolution should be chosen high enough to distinguish well between adjacent peaks in the spectrum.

The constraints for the frequency amplitudes are defined via a polygon based on a tabular parameter input. Typically, there are two kinds of parameterizations: Definition via absolute values for the constraints and definition in relation to the magnitude at a certain frequency. For relative definition, the user is requested for the respective base frequency. In case this frequency is not an integer multiple of the frequency resolution, the frequency closest to it is taken. With parameter *searchInterval* a search interval around this base frequency is defined, where the maximum peak in this region is taken as real base frequency. For example for the 50 Hz net frequency of the European power grid, the frequency may vary by $\pm 0.2 \text{ Hz}$ in regular operation mode. After initialization, the limits are displayed as red polygons in the icon

Whenever the Boolean input *condition* has a rising edge, the Real input signal *u* is periodically sampled with a sample rate automatically computed from f_{max} and f_{res} and stored in a buffer. Once “sufficient” values are stored in the buffer (for details, see below), an FFT is computed, displayed in the icon as bar plot and stored on file. Additionally, the distance

to the amplitude boundary is computed. If at least one amplitude is above the boundary, output $y = \text{Violated}$. If all amplitudes are below the boundary, $y = \text{Satisfied}$, and if the FFT has not yet been computed, $y = \text{Undecided}$. In the example of Figure 8, $y = \text{Satisfied}$.

In case a falling edge of u occurs before sufficient sample values are monitored or the simulation run is terminated, then the FFT spectrum is approximated via the partly-filled buffer with zeros for other values (called “zero-padding” technique).

Standard tools/functions for FFT provide a different, *user-unfriendly* parameterization. The mapping of the parameterization of the *WithinAbsoluteFFTDomain* block to the underlying standard FFT parameterization is non-trivial and is shortly sketched:

In order that the *amplitudes* are computed by the FFT with sufficient precision, the FFT computation needs to be performed for a much larger frequency as of interest for the user. In the block a fixed factor of 10 is used. So, if $f_{max} = 4$ Hz, then the FFT computation uses internally a maximum frequency $f_{max,FFT} \geq 40$ Hz. The basic formulae for an FFT computation of real numbers with even number of sample points are summarized in equation (1):

$$\begin{aligned} f_s &= \frac{n_s - 1}{T_s}, \\ f &= \left[0, \frac{f_s}{n_s}, \frac{2f_s}{n_s}, \dots, \frac{f_s}{2} \right], \\ \Delta u_r &= u(t_r) - u_{DC}, \\ n_f &= \frac{n_s}{2} + 1 \\ u_{FFT,k}(f_k) &= \frac{1}{n_f} \sum_{r=0}^{n_f-1} \Delta u_r e^{-i2\pi k \frac{r}{n_f}} \end{aligned} \quad (1)$$

where

- T_s is the sample period.
- n_s is the number of sample points
- f_s is the sample frequency ($f_{max,FFT} = \frac{f_s}{2}$)
- $\frac{f_s}{n_s}$ is the frequency resolution (f_{res}).
- n_f is the number of frequency points
- u_{DC} is the arithmetic mean of the signal
- Δu_r is the difference of the input signal with respect to the arithmetic mean u_{DC} .
- u_{FFT} is a complex number as function of a (real) frequency $f_k, k \in [1..n_s]$ and represents the FFT.

In order to be efficient, the original FFT algorithm by Cooley and Tukey (Cooley, 1965) requires that the number of sample points is an integer multiple of 2: $n_s = 2^i, i = 1, 2, \dots$. Newer algorithms allow more prime numbers. The implemented blocks use the public domain C-code KISS FFT (Borgerding, 2003). This mixed-radix FFT code requires that the number of sample points must be an integer multiple of 2, 3 and 5:

$n_s = 2^i 3^j 5^k$. For real signals, n_s must be additionally an *even* number.

The maximum frequency $10 \cdot f_{max}$ is now enlarged so that the number of sample points n_s fulfills the above restrictions. The sample period T_s is determined, so that the frequency resolution f_s/n_s has the required value. These computations are performed with the following Modelica code:

```
// Compute best ns according to 10*f_max and f_resolution
ns := 2*integer(ceil(10*f_max/f_res));

// Make ns even
ns := if mod(ns, 2) == 0 then ns else ns + 1;

// Find smallest ns that is even + expressed as 2^i*3^j*5^k
while true loop
  ns1 := ns;
  while mod(ns1, 2) == 0 loop ns1 := div(ns1, 2); end while;
  while mod(ns1, 3) == 0 loop ns1 := div(ns1, 3); end while;
  while mod(ns1, 5) == 0 loop ns1 := div(ns1, 5); end while;
  if ns1 <= 1 then break; end if;
  ns := ns + 2; // enlarge ns, but keep it even
end while;

// Compute other FFT variables
f_max_FFT = f_resolution*div(ns, 2);
Ts         = 1/(2*f_max_FFT) "Sample period";
T          = (ns - 1)*Ts     "Simulation time";
```

To understand the numbers above beforehand, utility function `showNumberOfFFTPoints(..)` is provided that computes them. For example calling the function as

```
showNumberOfFFTPoints(f_max=2000, f_resolution=27);
```

results in the following output:

```
Desired:
  f_max           = 2000 Hz
  f_resolution    = 27 Hz

Calculated:
  Maximum frequency used = 20250 Hz
  Number of sample points = 1500 (=2^2*3^1*5^3)
  Sample period         = 2.46914e-005 s
  Simulation time       = 0.0370123 s
```

Note, that

$$\begin{aligned} f_{max,FFT} &= (n_f - 1) \cdot f_{resolution} \\ &= \frac{n_s}{2} \cdot f_{resolution} \\ &= \frac{1500}{2} \cdot 27 \text{ Hz} \\ &= 20250 \text{ Hz} \end{aligned}$$

In the “advanced” tab access is given to parameters less often used:

- *SearchInterval* (search interval around base frequency)
- *TerminateAfterFFT* (When true, the simulation is terminated after evaluation of the FFT)
- Parameterization of the “Window” type

In case the sampled interval does not match a multiple length of the occurring waves, the spectrum would

suffer from this “discontinuity” of non-matching levels at start and end point since the FFT assumes periodic signals. This can be circumvented by multiplication of the time series by a filter of the same length, called “window function”. If this window function exhibits a shape with zero at start and end and some maximum in the middle, this discontinuity can be attenuated. By choice of a proper window function, erroneous high frequency signals will be diminished and the signal power at frequencies not precisely matched in the FFT output spectrum is smeared to the adjacent spectral points (called bins). For details see (Heinzel, 2002). The influence of windowing is demonstrated in Figure 10 and Figure 11. A sinusoidal signal of amplitude 1.5 and frequency 3.4 Hz is not matched by the FFT’s output resolution of 1 Hz. Figure 10 shows a peak at 3 Hz with an amplitude of 1.2, some amplitudes in the adjacent bins and content for all higher frequencies.

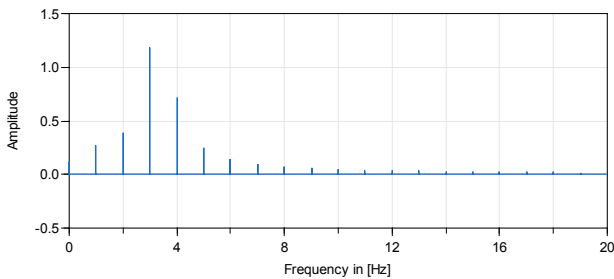


Figure 10: $u = 1.5 \cdot \sin(2\pi \cdot 3.4 \cdot t)$ and 1 Hz resolution.

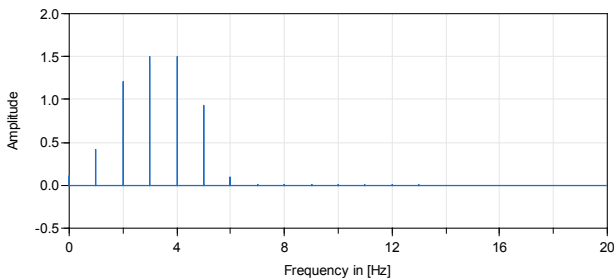


Figure 11: $u = 1.5 \cdot \sin(2\pi \cdot 3.4 \cdot t)$, 1 Hz resolution and flat top window.

In contrast, Figure 11 is the FFT output of the signal which was windowed by the “Minimum sidelobe 3-term-at top window SFT3M” (Heinzel 2002) of length n_s with the window

$$w_i = 0.28235 - 0.52105 \cdot \cos\left(1 \cdot 2 \cdot \pi \cdot \frac{i}{n_s - 1}\right) + 0.19659 \cdot \cos\left(2 \cdot 2 \cdot \pi \cdot \frac{i}{n_s - 1}\right), \quad (2)$$

$$i = 0..n_s - 1$$

One can see from the plot, both frequencies 3 Hz and 4 Hz show the amplitude of the original signal of 3.4 Hz. Also the next bins show a higher (erroneous) content while there are only low amplitudes at higher frequencies. As a consequence it is recommended to use windowing only in case where discrete peaks in the spectrum are expected, which may not be matched well

by the resolution, the output resolution is low and the information about the correct amplitude is essential.

In addition to the *WithinAbsoluteFFTdomain* block, the *WithinAbsoluteFFTdomain_THD*, calculates the Total Harmonic Distortion (THD) from the FFT output. THD is a measure for the amplitudes of harmonics in relation to the amplitude of the base frequency, where M is defined by $f_{base} \cdot M \leq f_{max,FFT}$:

$$THD = \sqrt{\sum_{k=2}^M A[k \cdot f_{base}]^2 / A[f_{base}]} \quad (3)$$

The THD criteria should only be evaluated for periodic steady state conditions. Periodic steady state is typically only occurring after an initial transient phase of the simulation. Instead of using an arbitrary settling time, the block offers the following feature: The THD can be evaluated cyclically at quite low numeric cost and is assumed to converge to a steady state value at periodic steady state condition. The *WithinAbsoluteFFTdomain_THD* block offers the option to evaluate the THD cycle every `update %` of the base harmonic until the difference between two successive THD evaluations is below `changerate`. At this point the criterion is calculated.

In Figure 12 some benchmarks for different kinds of data storage of the n_s FFT points is given:

- *SamplingAndModelicaBuffer* (= blue line) buffers the data at every sampling interval $1/f_s$ in a Modelica array. Due to Modelica’s single assignment rule, all values of this array need to get a value at every sample instant. If a value is not changed at the current sample instant, the value from the previous sample instant is copied (so at every sample instant $n_s - 1$ values are copied).
- *SamplingAndBuffer* (= red line) invokes a C function at every sample instant that stores the actual value of the input signal into an internal C array.
- *NoEventOnly* (= green line) does not use sampling but a C function stores the input value at every model evaluation into an internal C array. The values in this array are interpolated and internally sampled before the FFT is computed. For older Dymola versions this was beneficial since the simulation restart after a sample instant was “expensive” for a stiff solver. For newer Dymola versions this is not the case if the sampled system does not influence the integrator (which is the case here).

As can be seen from the figure *SamplingAndModelicaBuffer* is the slowest. *NoEventOnly* is a bit faster as *SamplingAndBuffer*. In other benchmarks, *SamplingAndBuffer* is the fastest approach. Due to these benchmarks, in the two blocks the *SamplingAndBuffer* approach is used for data storage.

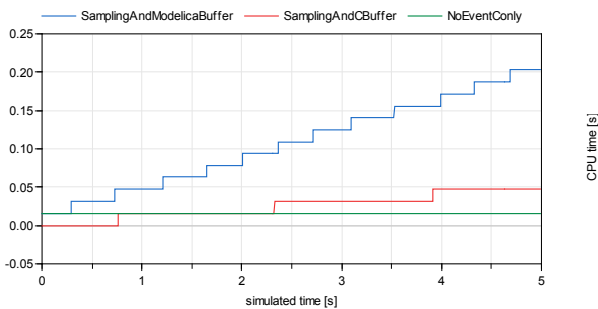


Figure 12: Comparison of CPU time [s] for three types of data storage for the FFT points.

6 Summary

In this paper the concept of model based specification and associated tools for aircraft systems was discussed. While previous work on this subject is based on MathWorks toolboxes it could be shown that Modelica could be used instead. Especially the new Modelica_Requirements library adds important extensions to express high level requirements and bind requirements to the system model under study. In combination with the FFT based requirement blocks of this paper, the full range of typical aircraft requirements for electrical systems can be formally defined. For automated documentation additional tools and scripts tailored to the need of the airframer or supplier is needed.

7 Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2016) for the Clean Sky Joint Technology Initiative under grant agreement no. CSJU-GAM-SGO-2008-001.

References

Becker C., and Giese T. (2011). Application of model based functional specification methods to environmental control systems engineering. *SAE Paper : Aerotech Congress & Exhibition*.

Becker C. et.al. (2013). Efficiency of model based methodologies in air systems engineering. *AST Workshop on Aircraft System Technologies*.

Becker C. (2014). Modellbasierter Entwurf von Flugzeugklimasystemen: Herausforderungen und Nutzen funktionaler Systemspezifikationen. *Technical report, Airbus Germany, EYVVC*.

Borgerding M. (2003). Kiss fft. URL: <http://sourceforge.net/projects/kissfft/>.

CleanSky (2014). Deliverable D2.1.4: Simulation and Design Platform Report. Revision b. *Technical report, Cleansky SGO*.

CleanSky project (2015). Systems for green operation (sgo). URL: <http://www.cleansky.eu>.

Cooley, James W.; Tukey, John W. (1965). "An algorithm for the machine calculation of complex Fourier series". *Math. Comput.* **19**: 297–301. doi:[10.2307/2003354](https://doi.org/10.2307/2003354)

Elmqvist H., Gaucher F., Mattsson S.E., and Dupont F (2012). State Machines in Modelica. *Proceedings of the 9th International Modelica Conference*, Munich, Germany, Sept. 3-5. Download: <http://www.ep.liu.se/ecp/076/003/ecp12076003.pdf>

Elmqvist H., Olsson H., and Otter M. (2015). Constructs for Meta Properties Modeling in Modelica. *Accepted for Modelica'2015 conference*.

G. Heinzel, A. Rüdiger and R. Schilling (2002). Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new at-top windows. URL: http://www.rssd.esa.int/SP/LISAPATHFINDER/docs/Data_Analysis/GH_FFT.pdf

Klößner A. (2014). The Modelica BehaviorTrees Library: Mission Planning in Continuous-Time for Unmanned Aircraft. *Proceedings of the 10th International Modelica Conference*, pp. 727–736, Lund, Sweden, March 10–12. DOI: 10.3384/ECP 14096727. Download: <http://www.ep.liu.se/ecp/096/076/ecp14096076.pdf>

Kuhn M.R. (2011). Advanced generator design using pareto-optimization. *Power Electronics and Drive Systems (PEDS)*, 2011 IEEE Ninth International Conference on, pp. 1061–1067, Dec. DOI: 10.1109/PEDS.2011.6147391.

Kuhn M.R., and Ji Y. (2014). Modelica for large scale aircraft electrical network V&V. *Proceedings of the 10th International Modelica Conference*, pp. 747-756. DOI 10.3384/ECP14096747. Download: <http://www.ep.liu.se/ecp/096/078/ecp14096078.pdf>

MathWorks (2015a). Simulink Toolbox: Verification and Validation. URL: <http://www.mathworks.com/products/simverification/>.

MathWorks (2015b). Stateflow. URL <http://www.mathworks.com/products/stateflow/>.

MIL704F (2004). MIL-STD-704F: Aircraft electric power characteristic. Download: http://everyspec.com/MIL-STD/MIL-STD-0700-0799/MIL-STD-704F_1083/

Myers T., Geoff Dromey R. and Fritzson P. (2010). Comodeling: From Requirements to an Integrated Software/Hardware Model. *IEEE Computer*, vol.44, no. 4, pp. 62-70, April 2011

Otter M., Thuy N., Bouskela D., Buffoni L., Elmqvist H., Fritzson P., Garro A., Jardin A., Olsson H., Payelleville M., Schamai W., Thomas E., Tundis A. (2015). Formal Modeling and Automatic Verification of Requirements. *Accepted for Modelica'2015 conference*.

Thuy N. (2014). D2.1.1 – Modelica extensions for properties modelling, Part III: FOrmal Requirements Modelling LAnguage (FORM-L). *Internal report, ITEA2 MODRIO project*, Sept. 2014.

Tunnat M. (2011). Integration modellbasierter Methoden in den Entwicklungsprozess hybrider Flugzeugregelungssysteme am Beispiel des Ventilation-Control-System. *Master thesis, Technical University Hamburg-Harburg, Institut für Flugzeug-Kabinensysteme, supervised by C. Becker and T. Giese (Airbus)*.

Multi Electrical Machine Pre-Design tool with error handling and machine specific advanced graphical design aid features based on Modelica

Tomasz D. Michaski¹ Antoni Garcia Espinosa² Jordi-Roger Riba Ruiz³ Luís Romeral Martínez⁴

^{2,3}Departament of Electrical Engineering, Universitat Politècnica de Catalunya, Spain,
garciae@ee.upc.edu, riba@ee.upc.edu

^{1,4}Departament of Electronic Engineering, Universitat Politècnica de Catalunya, Spain,
tomasz.michalski@mcia.upc.edu, luis.romeral@mcia.upc.edu

Abstract

This paper presents a design tool for Induction Machines, Permanent Magnet Synchronous Machines, Externally Excited Synchronous Machines and Switched Reluctance Machines. This software, based on Modelica language, is able to provide full dimensioning (cross and axial section measures) and operation characteristics according to mechanical and electrical requirements set as inputs. The tool is able to perform error handling, which informs a designer about unfeasible designs and gives clues about the possible errors. Both aspects of the tool GUI and scripts provide help files and code explanation in order to re-use the tool and improve library's functionalities.

Keywords: Modelica, design tools, electrical machines, SMPMSM, IPM, Synchronous Machine, SRM, Efficiency Map.

1 Introduction

Electrical motors use is spreading in new fields and they are replacing other actuators because of their performance, power, torque density and reliability. However, in order to integrate these new components into a system engineering design process, the models of electrical machines must be pre-evaluated and designed for specific uses.

In the past years several phenomes have changed the panorama regarding electrical machines typology. Regarding motors using magnets: appearance and proliferation of Surface Mounted Permanent Magnet Synchronous Motors, internal Permanent Magnet Motor Family, emerge and intensive development due to magnet's price raises (substitution of rare earth magnets by ferrite magnets) and increased power density requirements (keeping rare earth magnets in geometrical configurations meant to achieve high airgap flux). Regarding externally excited Synchronous Machines [1]-[2], its ability to work in four quadrants and to work under very high temperatures (where magnet motors get demagnetized) made them resurge in micro generation, both in land and aviation fields. These new appearances did not pull back Induction

Machines because of, as in Switched Reluctance Motors, its low construction and operation costs are still an advantage [3].

The electrical machine design process has also changed due to the reduction of FEA costs and high demand of motors for specific purposes. Both phenomena made it possible to start developing specifically optimized motors out of catalogue increasing the added value of small companies.

Not only design process has changed, but its implementation and tests before construction. The introduction of systems engineering design in Product Lifecycle Management requires parametrized models of all its components. Modelica not only fits in that spot but also allows multi-parametric, multi-physics implementation of such models for both: steady state and transient simulations [4].

The work presented in this paper is the first Modelica implementation of pre-design tool able to provide the data required to create such models in order to integrate the specific machine size and properties into system models and is intended to work under Clean Sky [5] European Initiative therefore the library along with help files, and manuals are free to use. This novel tool in the Modelica society also leaves a possibility to be used with other already developed open and commercial libraries i.e. outputs of pre-design sizing algorithms can be passed as inputs to electrical machines advanced and complex models such as the Actuation 2015 project [6].

In the cases where, in a system model, an electrical machine is required, the user is force to select from pre-designed models (with restricted electro-mechanical sizes) or manually calculate the machine parameters for the purpose he is modelling for. With this new tool, added to the Modelica chain, and given designer's specifications, insert the specific machine he needs to work with.

2 Basic Pre-design tool

2.1 Scope

Basic pre-design tool is conceived as a series of Modelica functions meant to return physical,

geometrical, electrical and magnetic properties as a starting point of an electrical machine's design. These scripts can be run from within Modelica or by a third party GUI able to call Modelica scripting. The program is able to pre-design Induction Motors (IM), Surface Mounted Permanent Magnet Synchronous Motors (SMPMSM), Internal Permanent Magnet Synchronous Motors (IPM) in their Spoke (embedded and non-embedded magnets), V-Shape and Planar configurations. It also performs pre-sizing for Switched Reluctance Motors (SRM) and externally exited Synchronous Machines (Syn). Each machine runs its own pre-sizing algorithm even though some of the sub-functions are shared across machines like winding factor calculation or number of gauged turns.

Since SMPMSM and IM machines have been deeply analyzed, this work is more focused in the IPM family, SRM and Synchronous machines.

For IPM family the program returns all cross section and axial section dimensions for stator, rotor, magnets shaft and hub if required. Following outputs are provided: moments of inertia of stator, rotor and total, mass of stator, rotor, total mass including housing as well as masses separated by materials like magnet mass, magnetic steel mass and copper. Subsequent are inductances and phase resistance based on desired working temperature. The tool also includes saturation factor of airgap flux, Back-EMF, number of turns per phase and final obtained power. With this data an efficiency map including losses is also calculated and performed.

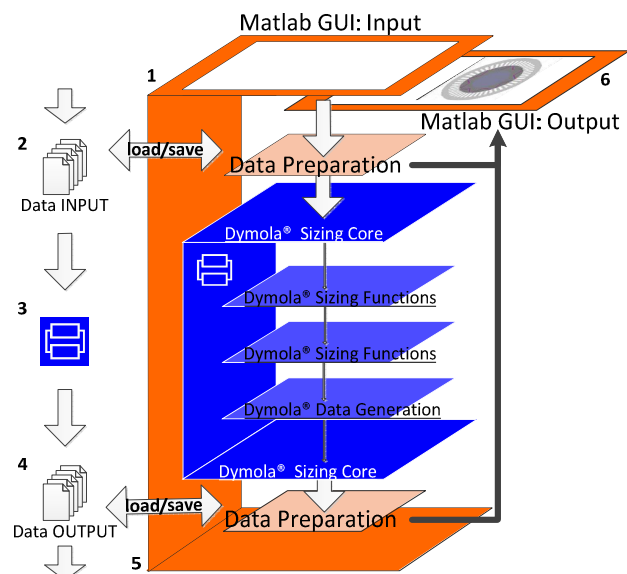


Figure 1. Layer structure of the GUI and Modelica pre-sizing core code.

2.2 Internal Operation and Process

The sets of inputs are large and vary from one machine to another. Among them there are first tier and second tier inputs. First tier are basic indispensable values for

pre-sizing, and second tier those pre-sizing tool can automatically set if left empty or its influence is minor. For the first tier inputs power, efficiency, current or voltage are some of the electrical parameters required. Airgap flux or magnetic steel or magnet properties are among the magnetic parameters. Part of the second tier inputs would be stator shoe tooth separation or shaft diameter external or extra length for housing. Because of the amount of inputs and its diversity an external input GUI was developed.

As Figure 1 shows, this GUI is a cover over Modelica functions. It allows opening and saving machines and has its own help documentation. When the inputs are set, the GUI prepares the data and sends it to Modelica by means of a callback of the sizing scripts which starts performing the sizing.

In this stage the several Modelica algorithms perform pre-sizing. When finished data is structured and send back to the GUI. In this stage is when motor can be saved with both input and output data. Also in this stage is when output data can be analyzed with the provided output sub tools.

2.3 Modelica Sizing Core

Modelica sizing core consists of five principal algorithms which perform sizing code for each machine: IM, PMSM, IPM, SRM and SYN. As shown in Figure 3, all of them utilize other common functions: winding (returns the winding factor of the fundamental frequency), $C_{mec}CALC$ (interpolates mechanical constant), $htrmodif$ (recalculates slot height based on trapezoidal slot to one based on squared slot), $hexapack$ (obtains a number of conductors per area following a hexagonal pattern), $quadrapack$ (obtains a number of conductors per area following a square pattern), $MassandMOI$ (calculates mass and moments of inertia), $Resistance$ (returns resistance per phase), $MassandMOISYN$ (calculates mass and moments of inertia of Synchronous machine), $round_Nph$ (rounds number of turns per phase), $cartdistance$ (returns distance between two points in Cartesian coordinates), $Rotation$ (performs basic algebraic rotation in Cartesian coordinates), $Reflection$ (performs basic algebraic reflection through axis situated at the angle position), $feasibleregion$ (determines if angle is feasible for SRM motor design), $TorquevsSpeed$ (returns torque vs speed vs efficiency map as a matrix), $Inductances$ (calculates d-q inductances), $RotorShape$ (generates a valid rotor geometry for IPM machine), $MAGNETdb$ (returns properties of selected permanent magnet) and $ellipse$ (finds ellipse from five points and returns its center).

Every main sizing function writes data in special format using `DataFiles`, `writeMATRIX` and `Utilities.Streams.print` functions. After that

machine structure is read by the external graphic user interface.

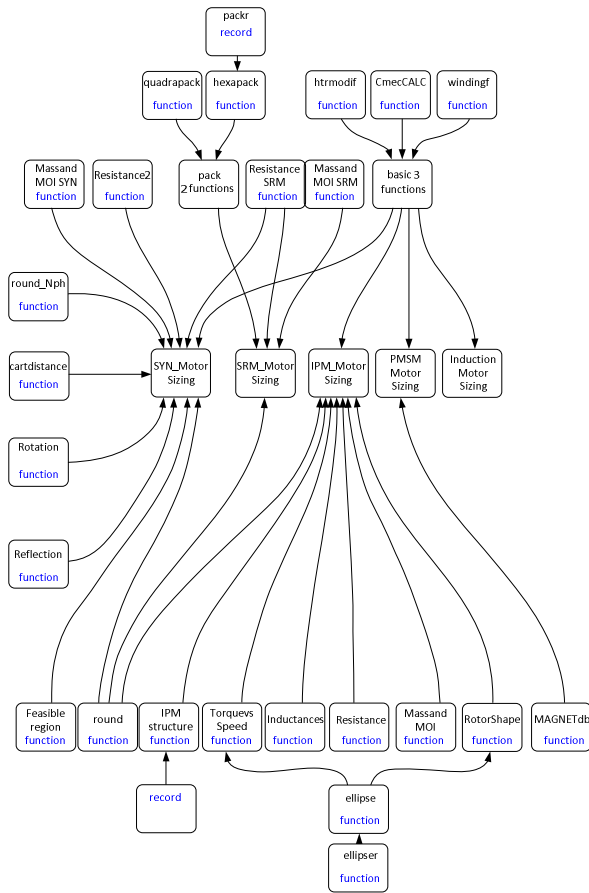


Figure 3. Modelica sizing relation tree of functions.

2.4 MATLAB GUI

Graphic User Interface was programmed in MATLAB Guide tool. This was done because it offers a good trade between complex plot representations and an appeal interface without the use of complex or third party APIs. All of the results from Dymola pre-sizing are decoded and interpreted on this stage.

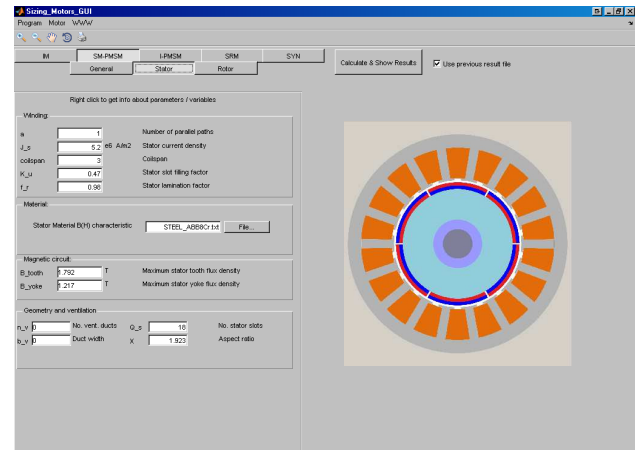


Figure 4. Pre-sizing GUI for SMPMSM inputs. Each variable contains a brief description when prompted.

The Graphical User Interface software is presented in Figure 2 and Figure 4. This was done because currently Dymola serves as a simulation environment and there are no tools publically available for generation of the graphical user interface [7]. However user is expected to performs various pre-design runs, inspect and print results if necessary.

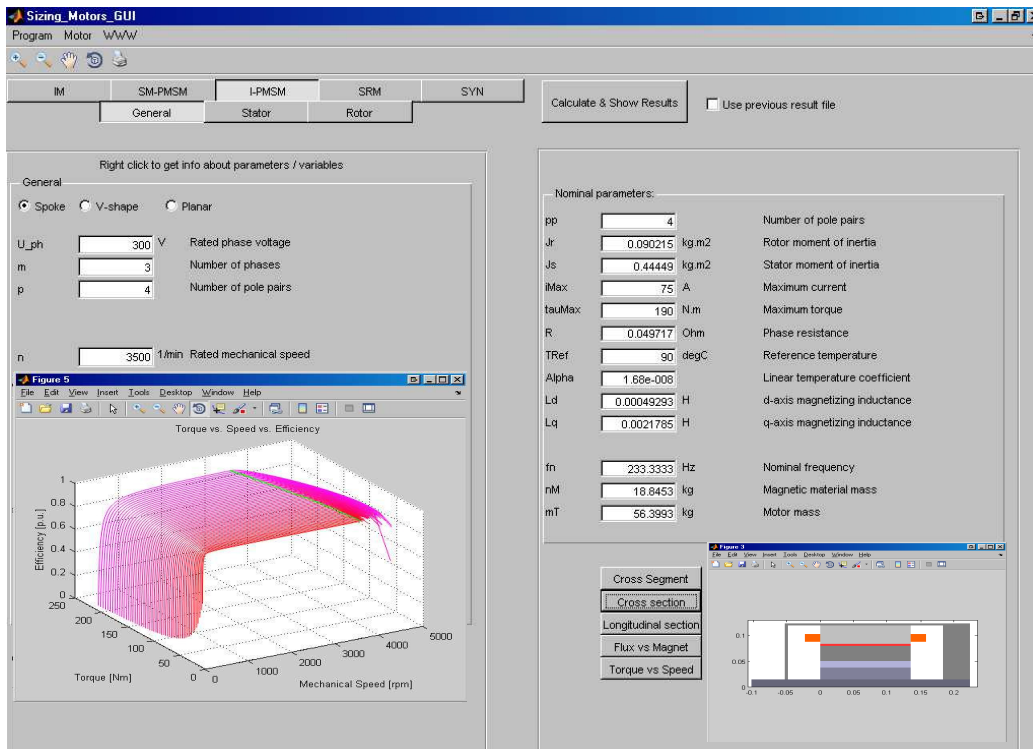


Figure 2. Example of Input/Output GUI with some windowed sub tools for the IPM machine.

3 Example of Use

V-shape Internal Permanent Magnet will be used in the example to show most functionalities of the tool including the graphic error handling, console messages and height vs. PM. Figure 5 shows Modelica's sizing core for this kind of machines.

When inputs are set and sent to sizing core several data is calculated before the actual sizing procedure starts, for example, if not set, shaft and hub diameters. Pole shoe height is determined depending on desired power.

And finally total length and polar pitches. Then stator sizing starts taking into account flux saturation by means of a flux form factor coefficient corrector. Desired fluxes in airgap, tooth and yoke and winding configuration will determine the final size, number of conductors and saturation.

Then rotor calculation starts. It is worth noting that IPM rotor can be very complex and incur in several parts collisions, this it is explained in more detail in section 3.2. That is why rotor is sized by two main loops, one for the geometrical and the other for the magnetic considerations. If rotor is physically impossible program aborts, if not, then geometry of magnet, its position and reluctance paths are send to the magnetic circuit solving algorithm that will determine if the magnet quantity is enough to sustain the airgap flux, if not, magnet height is increased and rotor data is send back to geometry validation step.

3.1 First input set

The desired phase voltage, phase current, power, rated torque, rated speed, power factor and efficiency (which later on are corrected). Number of phases, number of poles, form factor as well as winding properties are the main parameters for design process. During this initial round it is recommended to use default values for flux densities. Desired Airgap, Stator Tooth and Yoke Flux Densities are required, and, as second tier, geometric properties like shaft diameter and length, slot opening, or filling and stack factors. Although these parameters can have a great influence in machine's design, they have to adapt to electrical and mechanical requirements not the other way around.

Further iterations can be used to refine input parameters if output geometry or features doesn't look reasonable, for example big yokes or too narrow tooth tip. Then the script is launched and performs sizing. Usually a set of inputs leads to impossible geometry or magnetic circuit, which is why an error handling system was developed.

3.2 Error Handling: Console and Graphics

Internal PMSM incorporates very complex rotor geometry because of the number of independent elements that conforms it: rotor shaft, rotor hub, air barriers, wedge dimensioning and magnets size and

positioning [8]. Figure 6 shows geometry used to obtain geometry.

It is not suitable for an ease of use of pre-design to ask for more than twenty parameters associated only to those elements. The basic pre-design tool is able to position the magnet with four parameters, cover factor, desired wedge, top air barrier length and V-shape angle. Summed to the diameters automatically calculated magnets are positioned and all associated elements obtained (Figure 6): reluctance path lengths and widths, centers of gravity and areas. This is thanks to a parametrized model of its geometry.

Outfits and collisions are detected by rules of relative position of the aforesaid parametrization by means of

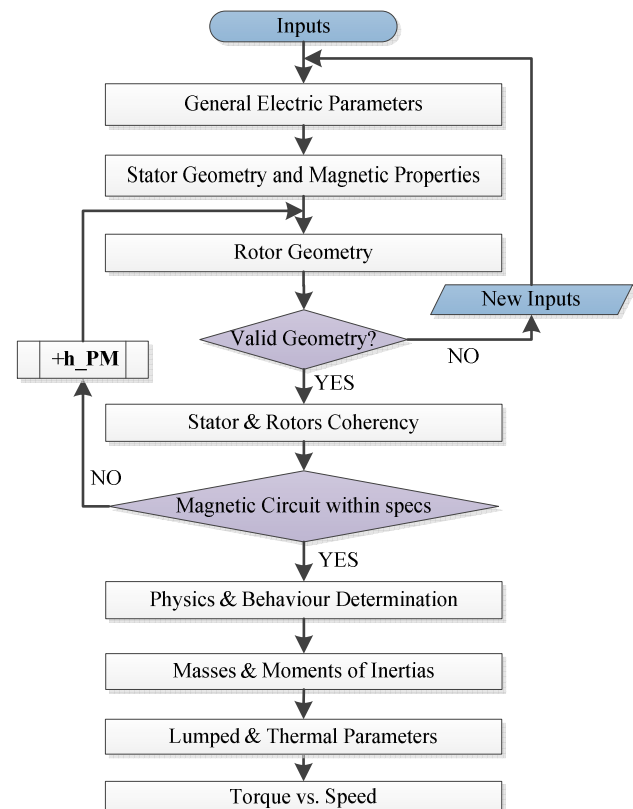


Figure 5. Sizing-core flow chart for IPM machines.

points. For example, α_v cannot be smaller than half pole pitch, which is the same to say that Point5 would be further than machine center.

Another typical example is when V-shape angle is too narrow, in that cases distance of Point4 is inferior than shaft radius which leads to a collision. The program is able to detect these collisions and report them in both graphical representation and console. Console messages appear at the very same moment that the problem is generated, for graphical representation, Modelica has to return the control to the GUI which will call the cross section function graphical representation.

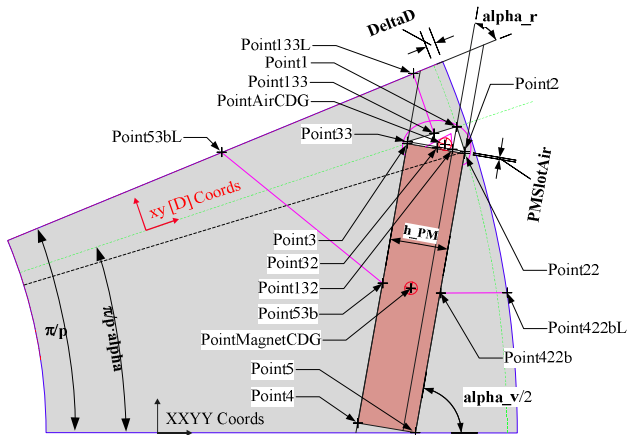


Figure 6. Semi pole graphical representation of the geometrical assistant points used to fit rectangular magnets inside rotor. This is only valid for V-shape with only one air barrier in top.

An example of the console messages are as follows:

```
**** ROTOR GEOMETRY CALCULATION FAILED,
because of geometry error:
***** ALLOCATION ERROR: Magnet may
penetrate the shaft area *****
*** Start: Flux-vs-h_PM GEN01 ***
*** End: Flux-vs-h_PM GEN01 *** [Success]
*** End IPM GEN 07 ***
```

Each geometry message errors return, when possible, the nature of the sizing error, as well, each function returns and START-END messages.

And the graphical representation is as shown in Figure 7. The plot is not rasterized; zooming to any area will not carry a loss of detail. Collision information is saved in motor structure allowing other third party applications to perform its own interpretation of collisions.

3.3 Final Iteration and results comparison

Problem with shaft diameter can be changed by two means, one increasing V-shape angle and the other reducing shaft diameter. The second solution becomes a better approach when it comes to reduce the impact in machine properties. Once enough space for magnets is given the final motor is generated.

By properly setting inputs it is possible to achieve close to FEA 2D results, this is shown in Table 1.

It is not the aim of the tool, and it cannot, to substitute FEA in design process. This level of basic pre-design does not allow taking into account complex reluctance network effects and saturation effects, which are directly related to d-q inductance values. However, being able to achieve these results it is especially useful when designing new machines between two well-known sizes or rated powers, or during optimization processes.

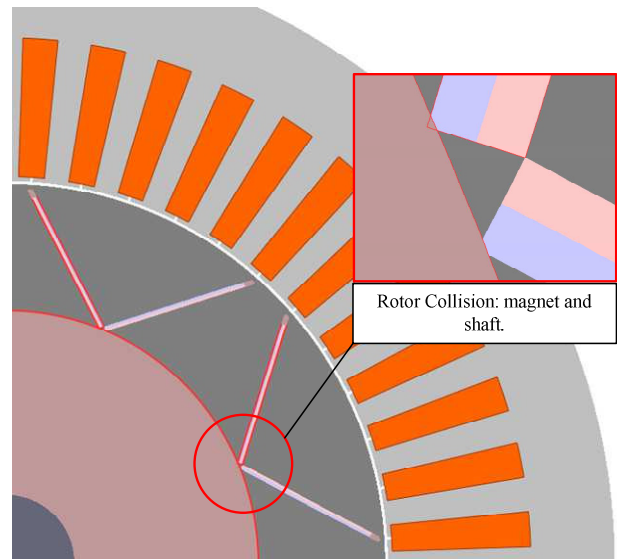


Figure 7. Graphical output error handling, remarks in red contours and dim colors of parts colliding. Zoom can be performed in figures to see the detailed problem.

Table 1. Comparison between FEA and pre-design too.

FEM/Real	Parameter	Pre-design Calculated	Relative Error
97,78	Total Mass [Kg]	100,35	2,6%
0,08	Rotor Moment of Inertia [Kg·m ²]	0,0832	2,7%
1,51	Stator Moment of Inertia [Kg·m ²]	1,504	0,2%
0,088	Phase Resistance [Ω]	0,0943	6,8%
0,0021	Direct Axis Inductance [H]	0,0026	25,4%
0,0091	Quadrature Axis Inductance [H]	0,01183	29,4%
0,0048	Final Flux in Airgap [Wb]	0,0044	8,0%
161,00	Back Electro Motive Force [V]	153,73	4,5%
48915,00	Rated Mechanical Power [W]	48939,56	0,1%
282,00	D outer stator [mm]	292,59	3,8%
184,00	D outer rotor [mm]	186,52	1,4%
75,00	Length [mm]	75,95	1,3%
30,00	Slot Height [mm]	30,72	2,4%
6,50	Tooth Width [mm]	6,05	7,0%
18,40	Yoke Height	21,72	18,0%

	[mm]		
27,00	Magnet Length [mm]	27,69	2,5%
5,00	Magnet Height [mm]	5,15	3,0%

3.4 Output sub-tools of the GUI

Raw data of the pre-design requires interpretation especially when the volume of information is high. In order to present pre-sizing outputs in an easy going way several tools were developed. Some are specific for each machine and some are shared across all packages.

All outputs depart from the output section of the GUI where basic sizing information is presented. An example of the output GUI is shown in Figure 8. Each machine has its own outputs sets, interface and tools.

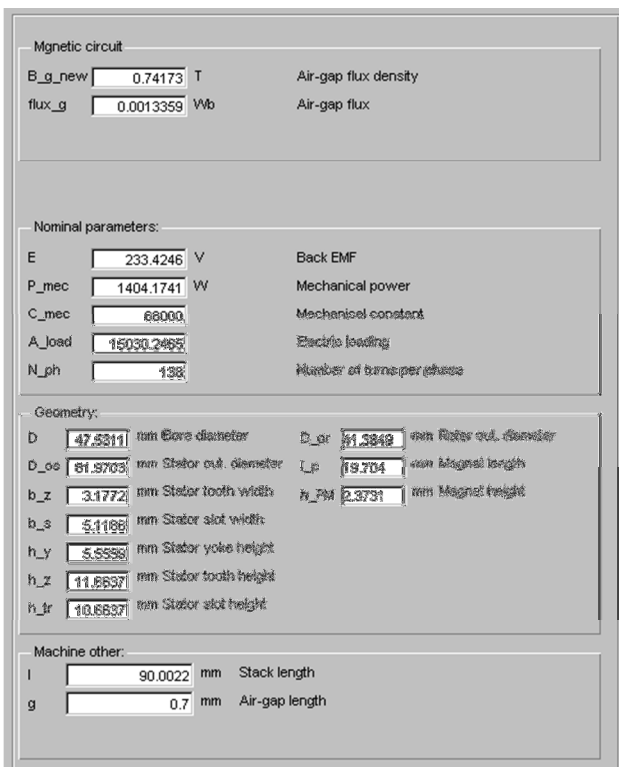


Figure 8. Screenshot of SM-PMSM pre-design outputs.

The most reliable and informative way to read dimensions relationship is to generate blueprints, but given dimension's magnitude are already computed, the use of scaled colored representations is more valuable. It is able to return fast qualitative information as well as raw dimensions. That is why scaled cross and axial sections of the machine were implemented in the output stage. Figure 9 contains two examples: for Synchronous Machine and Switched Reluctance Machine. This allows a fast eye check of dimensions and proportions giving almost instantly clues about

what can fail in a design. If the design is considered good the designer can directly import output values to his design table.

For SRM specific case a feasible region chart was elaborated [9]. An example of it can be seen in Figure 10. This chart locates the motor in a five conditions diagram. Three are related to the good behavior of the motor, and two to physical limitations given the desired number of poles for stator and rotor.

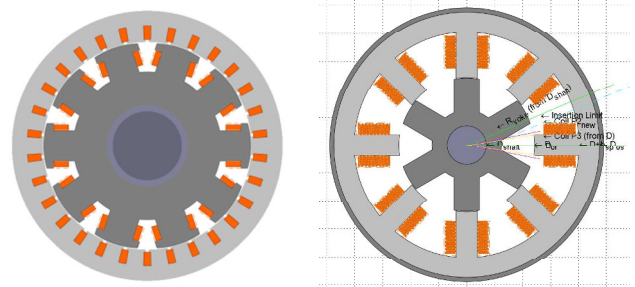


Figure 9. Example of two cross sections of different machines with different graphical detail set. Left, Salient Pole Synchronous Machine. Right, Switched Reluctance Motor with dimensional aids.

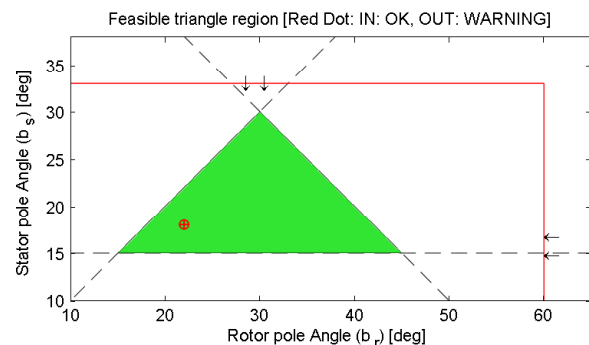


Figure 10. SRM Feasible Region Sizing Chart. Allows to determine if motor complies with 5 conditions between rotor and stator pole angles.

Also an efficiency map is generated as a preliminary representation of the motor behavior. Efficiency map is a representation of torque vs mechanical speed chart with efficiency of each point following a color gradient depending on its value (Figure 11). This map is able to take into account copper and iron losses. The map requires the following information from pre-design: inductances; phase voltage; field; average flux density and mass of each part where iron losses are considered (this is obtained thanks to the Moments of inertia and masses function); phase resistance and a maximum speed and current. The efficiency map computes, in a series of concatenated loops, the equivalent circuit of the machine taking into account both phase resistance in series and iron losses in a parallel branch. Iron losses are taken into account by means of an implementation of Bertotti's equation [10]. To calculate losses

following that method three elements are required: average field densities are found via the flux densities determined for each part, volumes (masses) are computed in a previous stage: Masses and Moments of Inertia function, and finally coefficients are found once material for each part has been selected or can be introduced manually.

4 Future work

The aim of any pre-design tool is to reduce the required information designer must provide in order to achieve a feasible design. This is directly related to the number of inputs required in the first iterations of the design, before entering into a refining process. On the other hand by means of further layers in the design pre-design tools can provide even richer sets of data without landing in the long simulation times of FEA solutions.

It should be able to handle several winding layers and fractional slotting and present the results graphically. One very complete free solution in this regard is EMETOR online SMPMSM tool [11].

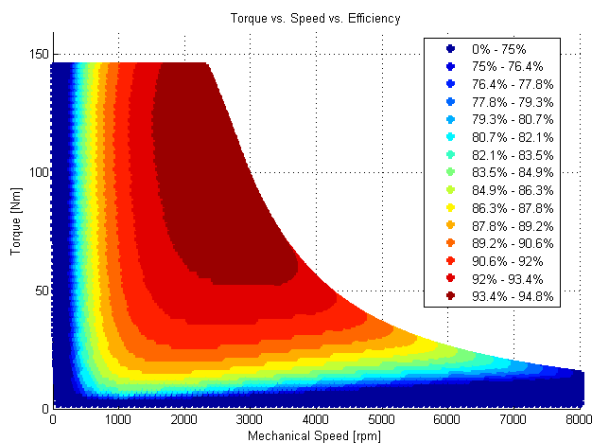


Figure 11. Torque vs. Speed vs. Efficiency representation (a.k.a. Efficiency map) for an IPM machine delivered by the pre-sizing tool. Data is calculated under Modelica, plotted by GUI.

Also the introduction of thermal restrictions by means of a thermal model would also optimize the design process and prepare the models for even more complex simulation scenarios. Thermal model should be able to interact with resistance values during pre-design process and also able to specify losses and heat transfer with the surroundings.

For the second proposed approach a reluctance network of each machine based on an electro-magnetic model of the machine would provide the golden mean between a basic pre-design and a FE analysis. Reluctance network should provide variables such as torque ripple frequencies and magnitudes, Back Electro Motive Force shape, airgap flux shape, both taking into account armature reaction. And several other variables

usually reserved to FE simulations. It also should be able to report variations in performance and behavior based on different control approaches given by Modelica implemented systems of such nature.

5 Conclusions

This tool provides an easy and intuitive way to generate machine designs in order to provide, to further design steps, seed values to work with. Also, by means of a refinement process, obtain close to final machine properties to use into system model design. It as well helps to determine pre-design inconsistencies and evaluate machine properties by means of the debug data in console, the graphical aids during design process and graphical representation of outputs.

Thanks to the inclusion of the efficiency map a preliminary behavior of the machine can also be evaluated before setting it up in any system.

Acknowledgements

Modelica implementation of this pre-design tool is part of Modelica Library of Detailed Magnetic Effects in Rotating Machinery (MAGMOLIB, SP1-JT1-CS-2013-01, GA-620087) project managed by the German Aerospace Center (DLR) and developed by MClA research center a part of the CLEANSKY partnership, a Public Private Partnership between the European Commission and the aeronautical industry.

References

- [1] M. R. Kuhn, A. Griffo, W. Jiabin, and J. Bals, "A components library for simulation and analysis of aircraft electrical power systems using Modelica," in *Power Electronics and Applications, 2009. EPE '09. 13th European Conference on*, 2009, pp. 1-10.
- [2] M. R. Kuhn, "Advanced generator design using pareto-optimization," in *Power Electronics and Drive Systems (PEDS), 2011 IEEE Ninth International Conference on*, 2011, pp. 1061-1067.
- [3] J. Yang, "A novel modelica based design platform for switched reluctance drive systems," in *Electrical Machines and Systems (ICEMS), 2014 17th International Conference on*, 2014, pp. 3302-3308.
- [4] C. Kral, A. Haumer, and R. Wöhrnschimmel, "Extension of the FundamentalWave Library towards Multi Phase Electric Machine Models," ed, 2014.
- [5] *The Clean Sky Joint Technology Initiative*. Available: <http://www.cleansky.eu/>
- [6] (2015). *Actuaction2015*. Available: <http://www.actuaction2015.eu/>
- [7] C. Schlegel and R. Finsterwalder, "Automatic Generation of Graphical User Interfaces for Simulation of Modelica Models," presented at the Modelica Conference 2001, 2011.
- [8] J. R. a. M. Hendershot, T.J.E., *Design of brushless permanent-magnet motors*. Hillsboro, OH : Oxford : Magna Physics Pub: Motor Design Books LLC, 1994.
- [9] R. Jordi-Roger, G. Antoni, and R. Luis, "A computer experiment to simulate the dynamic behaviour of

- electric vehicles driven by switched reluctance motors," vol. 51, ed: International Journal of Electrical Engineering Education, 2014, pp. 368-382.
- [10] M. Chunting, G. R. Slemon, and R. Bonert, "Modeling of iron losses of permanent-magnet synchronous motors," *Industry Applications, IEEE Transactions on*, vol. 39, pp. 734-742, 2003.
- [11] R. I. o. Technology. (2008). *EMETOR*. Available: www.emc.ee.kth.se/emetor

Enhancements of Electric Machine Models: The EMachines Library

Anton Haumer^{1,2} Christian Kral²

¹OTH Regensburg, Germany, anton.haumer@oth-regensburg.de

²EDrives, Austria, {anton.haumer, christian.kral}@edrives.eu

Abstract

Transient models of multi phase electric machines are already implemented in the Modelica Standard Library (MSL). However, advanced effects like saturation and skin effect are not taken into account. As an extension to the MSL models, the new EMachines library is presented. This package will be released as a supplemental library to the commercial EDrives library. The particular focus of this paper is on the deep bar effect of induction machines. A comparison of simulation results demonstrates the influence of the skin effect on the operational behavior of the machines. At the end of this publication further developments of the EMachines library will be outlined.

Keywords: multi phase electric machines, induction machines, squirrel cage, deep bar effect, skin effect

1 Introduction

The Modelica Standard Library already contains transient models of multi phase electric machines: `Modelica.Electrical.Machines` and `Modelica.Magnetic.FundamentalWave` (Kral, Haumer, Wöhrnschimmel, 2014). For the next release of the MSL, quasi static machine models are planned to be included. These models neglect electric transients for performance reasons (Kral, Haumer, 2014):

`Modelica.Magnetic.QuasiStaticFundamentalWave`.

Both the transient and the quasi static models consider Joule, friction, core and stray load losses. However, more advanced effects like saturation and skin effect are not taken into account. During the development of the EDrives Library (Haumer, Kral, 2014) and the extension towards controlled multiphase operation, the stator stray inductance $L_{s\sigma}$ design had to be changed. Therefore, in the EMachines library the stator stray inductances are modeled by phase inductances with mutual coupling to other phases and by self inductances – ideally coupled with the respective phase only. The new leakage inductance concept was implemented in the electric machine models of the EMachines library without affecting the backwards compatibility of the machine parameters. Yet the new machine models are based on a parameter record to overcome the drawback of parameterization

by multiple parameters. In the new EMachines Library advanced effects like saturation and skin effect will be taken into account. First, the deep bar effect is implemented, followed by further effects.

The EMachines library will be released as a supplemental library to the commercial EDrives library (see Table 1). Yet the EMachines library can be used without the EDrives library to investigate the influence of advanced effects of machines fed directly from the supply network.

Table 1. Description of electric drives libraries

library	description
Modelica. Electrical. Machines	transient threephase models, based on space phasor theory, connector and parameter compatible with <code>Magnetic.FundamentalWave</code>
Modelica. Magnetic. FundamentalWave	transient multiphase models, based on coupling between electrical domain and fundamental magnetic field
Modelica. Magnetic. QuasiStatic. FundamentalWave	quasi static multiphase models, based on coupling between electrical domain and fundamental magnetic field, neglecting electrical transients based on time phasors
EMachines	supplement to the commercial EDrives library, extending <code>Magnetic.FundamentalWave</code> and <code>Magnetic.QuasiStatic.FundamentalWave</code>
EDrives	commercial library for inverter fed drives, utilizing models from EMachines

First, the structure of the EMachines library (see Figure 1) will be presented, including the parameterization by means of parameter records. Second, the technical details of the deep bar effect will be explained. Comparisons of simulation results demonstrate the handling of the machine models and the influence of the skin effect. At the end of this paper an overview on the future developments of the EMachines library will be given.

2 Structure of the Library

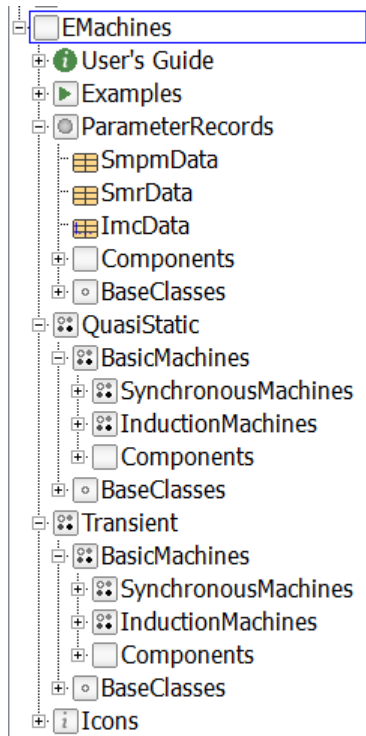


Figure 1. Structure of the EMachines Library

The EMachines library includes both transient and quasi static models of synchronous and induction machines. The EMachines models reuse the components of the Modelica.Magnetic.FundamentalWave and the Modelica.Magnetic.QuasiStaticFundamentalWave library and are thus fully connector compatible. Even though the parameters of the MSL library are also used by the EMachines models, the EMachines library is not parameter compatible since a parameter record is used (see Figure 2). This concept allows the user to switch from the parameter set of one drive to another parameter set in a very convenient way. All parameters that do not specify the machine properties but operational conditions remain single parameters in the machine models (see Figure 3). These single parameters are operational temperatures in case of a disabled heat port of the machine, and the enabling parameter of the optional support flange.

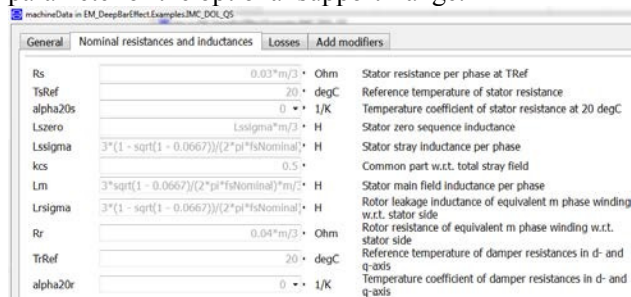


Figure 2. Parameter record of the induction machine

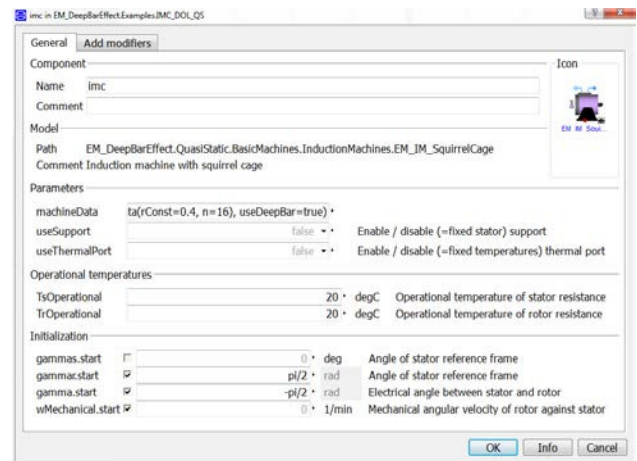


Figure 3. Operational parameters of induction machine

Components for modeling enhanced effects are stored in sub-packages named Components. The parameter records of these components and the entire machines are defined in the package ParameterRecords. Extensions are implemented in such a way that the user can switch them off or on. This concept enables the convenient comparison of the operational behavior of standard and enhanced electric machine models.

The EDrives library uses wrapper models. In these wrapper models the instances of EMachines models are used, including temperature and rotor position sensors. The wrapper models also use a signal bus connector to exchange the sensed quantities with the inverter models (Haumer, Kral, 2014).

Up to now, the following machine types were implemented:

- Synchronous machine with permanent magnets
- Synchronous reluctance machine and
- Induction machine with squirrel cage rotor.

Further machine types (electrically excited synchronous machine, induction machine with wound rotor and slip rings) will be implemented in the near future.

3 Deep Bar Effect

The skin effect in the bars of the squirrel cages of induction machines and the damper cages of synchronous machines is called deep bar effect (Binder, 2012; Toliyat, Kliman, 2004). Skin effect in general is caused by the linkage between electric and magnetic field, it describes the fact that current density increases from the center of a conductor to its surface, dependent on the current's frequency. In this paper it is assumed that the bar width is smaller than the height. Considering the distribution of the stray field (Kleinrath, 1975), it is sufficient to only consider a one dimensional model of the skin effect. The deep bar effect increases the effective resistance and decreases the effective stray inductance of the rotor bars. This effect strongly depends on the electrical rotor

frequency. Therefore the stator current and torque of induction machines at stand still, i.e. slip = 1, is strongly affected. Furthermore, the additional losses caused by higher harmonics of non-sinusoidal currents due to inverter operation are increased. Modeling the deep bar effect allows the investigation of the starting behavior of induction machines fed by the grid in a more realistic way. For inverter fed machines, the influence of resistance increase and stray inductance decrease on the inverter's controller can be investigated.

3.1 Differential equations

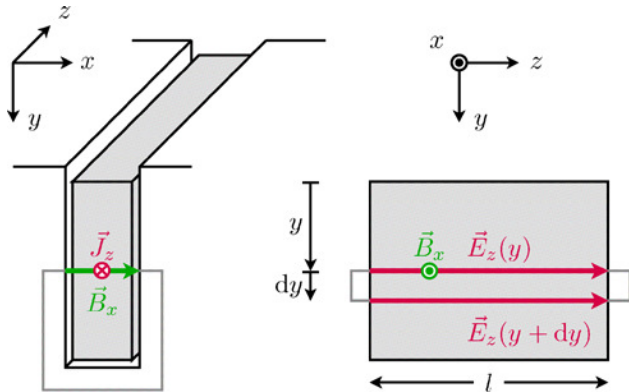


Figure 4. Deep bar embedded in rotor sheets

In Figure 4, one can relate the magnetic stray field strength H_x , and the magnetic flux density B_x ,

$$\vec{B}_x = \mu_0 \vec{H}_x \quad (1)$$

respectively, with the current density J_z . The magnetic permeability of the iron sheets is considered to be infinitely high:

$$\vec{H}_x(y)b(y) = \int_y^h \vec{J}_z(y')b(y')dy' \quad (2)$$

where h designates the total height of the bar.

The magnetic flux in x-direction of a layer of infinitely small height can be expressed as

$$d\phi = \mu_0 H_x(y)l dy = \mu_0 \frac{ldy}{b(y)} \int_y^h J_z(y')b(y')dy' \quad (3)$$

where l is the conductor length. If the bar current and therefore the bar current density is varying with respect to time, the time varying flux induces a voltage in the bar in z-direction, which in turn influences the current flow

$$\oint \vec{E} \cdot d\vec{s} = [E_z(y+dy) - E_z(y)]l = -\frac{\partial(d\phi)}{\partial t} \quad (4)$$

$$\kappa E_z(y) = J_z(y) \quad (5)$$

In (5) κ represents the electric resistivity. The current flow caused by the induced voltage reduces the current density for greater y (i.e. towards the slot ground) and increases the current density for smaller y

values (i.e. towards the slot opening). The partial differential equation for the transient skin effect yields:

$$\frac{\partial^2 E_z}{\partial y^2} = \mu_0 \kappa \frac{\partial E_z}{\partial t} \quad (6)$$

Assuming a sinusoidal current with constant frequency, and utilizing space phasors to express the sinusoidal quantities (Haumer *et al.*, 2008), the partial differential equation simplifies to an ordinary differential equation with respect to the height coordinate y :

$$\frac{d^2 \underline{E}_z}{dy^2} = j\omega\mu_0\kappa \underline{E}_z \quad (7)$$

For a rectangular bar the solution of (7) leads to an exponential distribution of the current density:

$$J_z = J_0 e^{-(1+j)\frac{y}{\delta}} \quad (8)$$

where $\delta = \sqrt{\frac{2}{\omega\mu_0\kappa}}$ denotes the skin depth and J_0 represents the current density at the top of the bar.

Considering a rectangular bar made of aluminum with a height of 30 mm and a width of 4 mm, the current densities with respect to height are depicted in Figure 5 for three different frequencies. The total bar current in all cases gives 120 A.

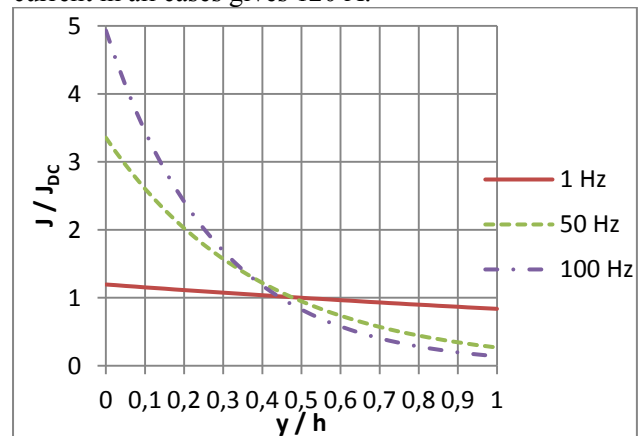


Figure 5. Current densities with respect to height

For simple geometries like rectangular or trapezoidal bars (zur Nieden, 1931), the differential equation (7) can be solved analytically. It is then possible to express the frequency dependent factors of resistance increase, k_R , and stray inductance decrease, k_L , with respect to the DC resistance and leakage inductance, respectively. For bars with rectangular shape, these factors are well known as Field's formulas, utilizing the so-called reduced height of the bar ξ :

$$\xi(\omega) = h\sqrt{\omega\frac{\mu_0\kappa}{2}} \quad (9)$$

$$k_R = \xi \frac{\sinh(2\xi) + \sin(2\xi)}{\cosh(2\xi) - \cos(2\xi)} \quad (10)$$

$$k_L = \frac{3}{2\xi} \frac{\sinh(2\xi) - \sin(2\xi)}{\cosh(2\xi) - \cos(2\xi)} \quad (11)$$

To cope with arbitrary bar shapes, the height of the bar is discretized as described in (Müller *et al*, 2008). The rotor resistance R_R and rotor stray inductance $L_{R\sigma}$ are separated into the constant parts $R_{R,con}$ and $L_{R\sigma,con}$ not affected by the skin effect and the variable parts $R_{R,var}$ and $L_{R\sigma,var}$ of the bar. The series connection of the variable part $R_{R,var}$ and $L_{R\sigma,var}$ is either replaced by a ladder network where each L-R element represents a layer of the discretized height, or the algorithm described in section 3.3 is evaluated. However, this algorithm can be applied only for quasi static points of operation: for transient operation, (6) has to be solved, using a ladder network (section 3.4).

The original cage used in `Modelica.Magnetic.FundamentalWave` and `Modelica.Magnetic.QuasiStaticFundamentalWave` is replaced by a new cage model shown in Figure 6. The sub-model `deepBar` represents the variable parts of the rotor resistance $R_{R,var}$ and stray inductance $L_{R\sigma,var}$ under influence of the deep bar effect, whereas the constant parts of the rotor resistance and stray inductance is represented by the single components `resistorConst` and `inductorConst`. For performance reasons, the quasi static cage model utilizes the algorithm described in section 3.3, whereas the transient cage model uses the ladder network of section 3.4. The usage of the quasi static algorithm increases performance due to the fact that the algebraic equations can be solved easier than the set of differential equations describing the ladder network.

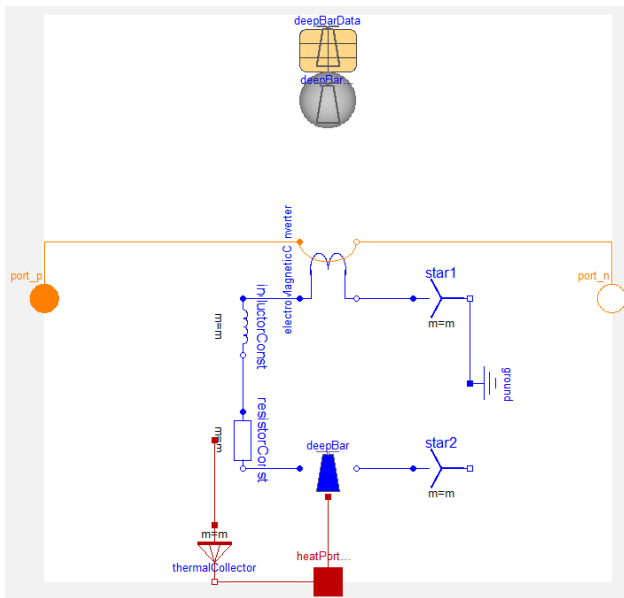


Figure 6. Transient cage model with skin effect

3.2 Parameterization of the Cage

For both – the quasi static algorithm and the transient ladder network – the resistances and stray inductances of the layers are pre-calculated in the parameter record of the cage: the user has to define the geometry of the

bar (i.e. a matrix with the height coordinate y in the first column and the respective bar width in the second column). An algorithm calculates the height h_k and the width b_k of each of the n layers. From these heights and widths, the resistances R_k at reference temperature and inductances L_k of each layer are determined:

$$R_k = \frac{l}{\kappa} \frac{1}{h_k b_k} \quad (12)$$

$$L_k = \mu_0 l \frac{h_k}{b_k} \quad (13)$$

For the actual resistances the operating temperature has to be considered.

The user has to define the constant part $R_{R,con}$ of the rotor resistance with respect to the stator winding. Thus the variable part $R_{R,var} = R_R - R_{R,con}$ is determined, too. Since the geometry of the bar defines the area of cross section and therefore the DC resistance of a single bar, R_{bar} , the turns ratio between stator winding and a rotor bar is given:

$$turnsRatio^2 = \frac{R_{R,var}}{R_{bar}} \quad (14)$$

The geometry defines the DC slot stray inductance L_{bar} , too. Therefore the variable and the constant, $L_{R\sigma,var}$ and $L_{R\sigma,con}$, respectively, are determined by:

$$L_{R\sigma,var} = turnsRatio^2 L_{bar} \quad (15)$$

$$L_{R\sigma,con} = L_{R\sigma} - L_{R\sigma,var} \quad (16)$$

3.3 Quasi static Algorithm

In order to determine the actual resistance and stray inductance, we may assume the complex current in the first layer at the bottom of the bar, and recursively calculate one layer current after another, utilizing the Modelica definition of complex numbers:

```

Modelica.SIunits.ComplexCurrent ik[n]
"Layer currents";
Modelica.SIunits.ComplexCurrent iSum[n]
"Summed layer currents";

Modelica.SIunits.Resistance Ractual
"Actual resistance";
Modelica.SIunits.Inductance Lactual
"Actual inductance";

equation
ik = {if k == 1 then Complex(1, 0) else
      (RRef[k - 1]/RRef[k]*ik[k - 1] + j*omega*
       Lsigma[k - 1]/(kT*RRef[k])*iSum[k - 1])
      for k in 1:n};
iSum = {if k == 1 then ik[1] else
        (iSum[k - 1] + ik[k]) for k in 1:n};
Ractual = sum({kT*RRef[k]*('abs'(ik[k]))^2
              for k in 1:n})/('abs'(iSum[n]))^2;
Lactual = sum({Lsigma[k]*('abs'(iSum[k]))^2
              for k in 1:n})/('abs'(iSum[n]))^2;
    
```

In this algorithm $RRef[k]$ represents the reference resistance of each layer and kT is the relative resistance increase at operating temperature compared to reference temperature. The total resistance and inductance calculations are based on the linear equivalent circuit shown in Figure 7.

Based on the actual resistance and stray inductance the actual voltage drop and Joule losses of the bar are calculated.

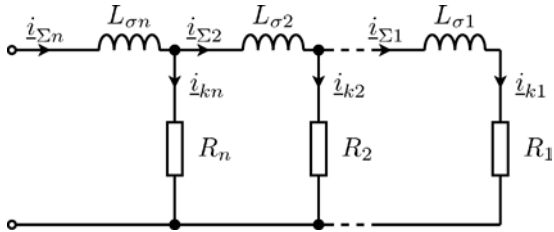


Figure 7. Equivalent circuit of a rotor bar

3.4 Transient Equivalent Ladder Network

In the transient deepBar model, a physical series connection of n (count of layers) R-L elements is established; The entire network structure is depicted in Figure 7 and a single R-L element is shown in Figure 8. The single R-L elements are connected in series.

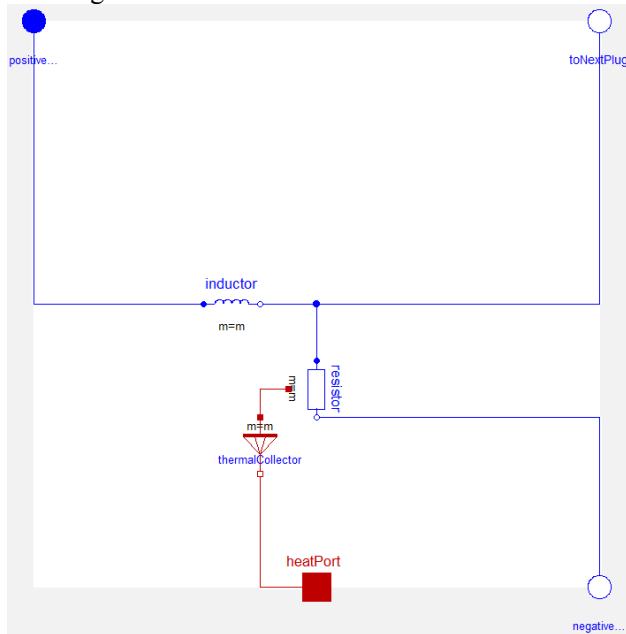


Figure 8. Transient deepBar model

4 Simulation Results

First, we compare the frequency dependent resistance increase k_R and stray inductance decrease k_L for a trapezoidal bar. For this bar shape the quasi static and the transient simulation and the analytically derived result (zur Nieden, 1931) are compared.

Figure 9 shows the quasi static model to evaluate the deep bar algorithm. The transient model looks similar; the frequency ramp raises very slowly from nearly 0 to 100 Hz to avoid transient effects as far as possible. Current and voltage are measured to calculate the effective impedance. The parameters of the investigated bar are summarized in Table 2. The bar is discretized into 16 layers.

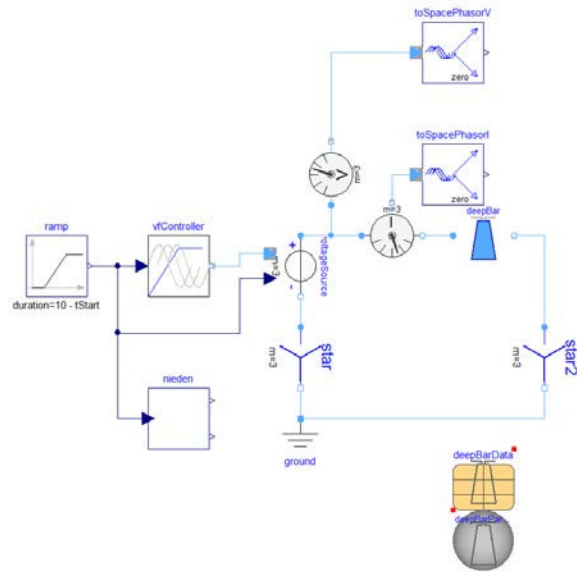


Figure 9. Quasi static evaluation of deep bar algorithm

Table 2. Parameters of the investigated bar

height of bar	30	mm
top width	6	mm
bottom width	2	mm
conductivity	$36 \cdot 10^6$	S/m at 20°C

Both the quasi static and the transient model calculate the resistance increase, k_R , and stray inductance decrease, k_L , from the actual Joule losses and energy of the magnetic field. The results of these calculations are summarized in Table 3 and Figure 10.

The transient model shows initial transients caused by the layer currents penetrating the bar. However, both the quasi static and the transient model match very well. Since the number of layers was chosen relatively low, there are deviations from the analytically obtained results. An appropriate choice of the count of layers leads to a trade-off between performance and accuracy of the results.

Table 3. Resistance increase and inductance decrease

f / Hz	analytical		quasi static		transient	
	kR	kX	kR	kX	kR	kX
1	1,001	1,000	1,001	1,000	1,000	0,882
10	1,072	0,969	1,080	0,969	1,075	0,964
20	1,266	0,891	1,276	0,894	1,273	0,894
30	1,505	0,797	1,509	0,808	1,508	0,808
40	1,731	0,711	1,731	0,730	1,730	0,730
50	1,929	0,640	1,926	0,667	1,925	0,667
60	2,100	0,584	2,095	0,618	2,095	0,618
70	2,251	0,540	2,245	0,579	2,245	0,579
80	2,388	0,504	2,381	0,547	2,380	0,547
90	2,515	0,475	2,506	0,522	2,506	0,522
100	2,635	0,451	2,623	0,500	2,623	0,500

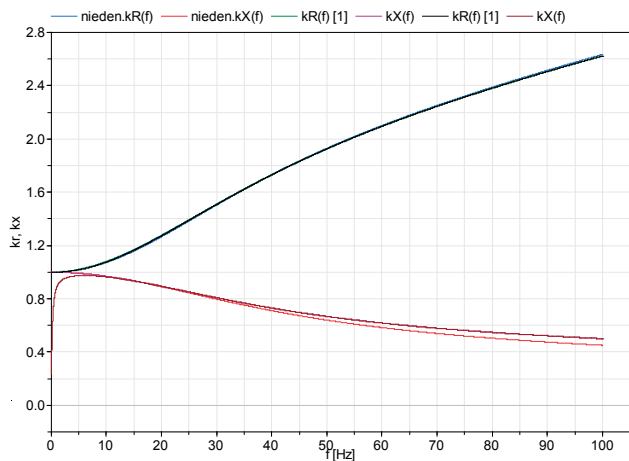


Figure 10. Quasi static (green and magenta) and transient (black and brown) results, compared with analytical formulae (blue and red)

The second example is derived from `Modelica.Electrical.Machines.Examples.AsynchronousInductionMachines.AIMC_DOL` and simulates the start-up of an induction machine with squirrel cage, both quasi static and transient. Both models have the same diagram layout; the transient model is shown in Figure 11. Each induction machine model is loaded with a quadratic speed dependent torque.

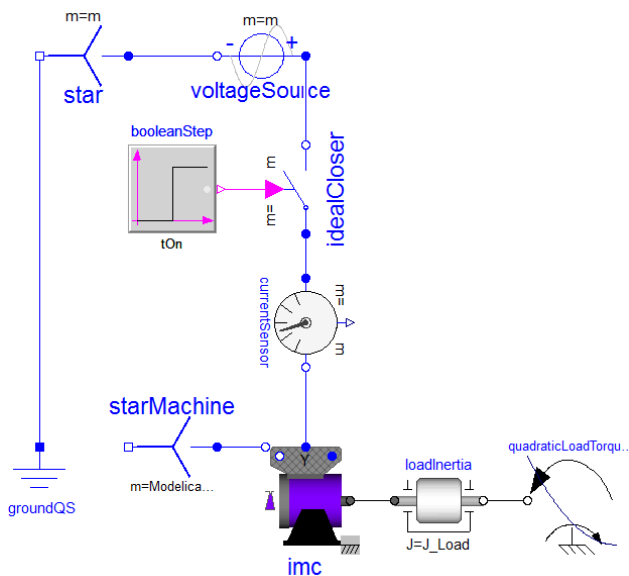


Figure 11. Transient start-up of induction machine

For the investigated machine the default data of the MSL induction machines are used; for the bar, the dimensions described in Table 2 are utilized.

Figure 12 shows the electromagnetic torque without and with deep bar effect of a quasi static simulation. Figure 13 shows the electromagnetic torque without and with deep bar effect of a transient simulation. Obviously, the deep bar effect causes a higher torque and reduces the duration of acceleration.

In each of the investigated cases the equilibrium at the end of the acceleration is practically the same, since

for small slip (i.e. rotor frequency) the skin effect can be neglected.

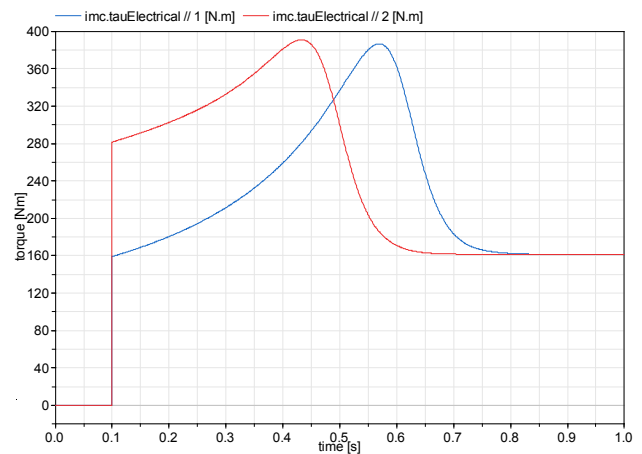


Figure 12. Quasi static torque w/o (blue) and with (red) skin effect

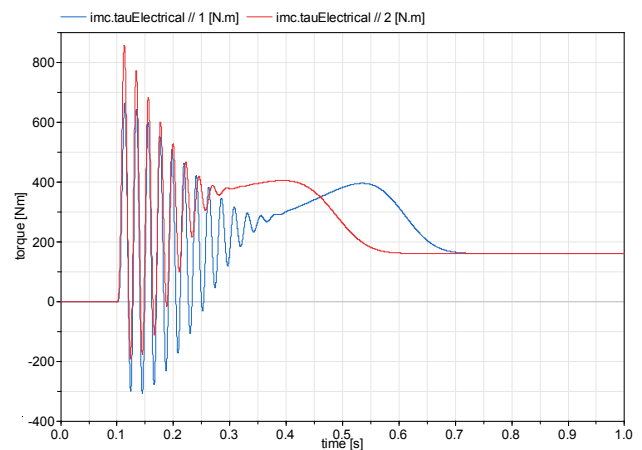


Figure 13. Transient torque w/o (blue) and with (red) skin effect

5 Conclusions and Outlook

The new EMachines library will be released as a supplemental library to the commercial EDrives library. The advantages and extensions to MSL models have been presented, especially the convenient parameterization of machine models by means of a parameter record.

In detail, the skin effect and its implementation have been discussed. A comparison of numerical and analytical factors of quasi static resistance increase and inductance decrease show satisfying coincidence. An example demonstrates the influence of the deep bar effect on the starting behavior of an induction machine.

All models are documented thoroughly and have been tested with both OpenModelica and Dymola. Examples demonstrating the usage are available.

The EMachines library is a perfect platform for extensions of machine models in the near future without losing backwards compatibility to the MSL. Further developments are already planned:

- Induction machine models with wound rotor and slip rings
- Electrically excited synchronous machines
- Saturation effect of the main field and of the stray field
- Thermal models of electric machines
- Temperature dependent characteristic of permanent magnet
- Effect of cross-coupling between d- and q-axis in synchronous machines
- Effects of higher harmonics of the spatial distribution of the magnetic field
- Hysteresis core losses
- Detailed losses in the permanent magnet of permanent magnet synchronous machines
- Coupling with FEA software in order to take into account the detailed magnetic operation point

References

- Andreas Binder. *Elektrische Maschinen und Antriebe*. Springer, 2012. doi: 10.1007/978-3-540-71850-5
- H. A. Toliyat, G. B. Kliman. *Handbook of Electrical Motors*. CRC Press, 2004. ISBN 978-0824741051
- Hans Kleinrath. *Grundlagen elektrischer Maschinen*. Akademische Verlagsgesellschaft 1975. ISBN 3-400-00279-8 (out of print).
- Anton Haumer, Christian Kral, Johannes Vinzenz Gragger, Hansjörg Kapeller. *Quasi-Stationary Modeling and Simulation of Electrical Circuits using Complex Phasors*. Modelica 2008.
- Anton Haumer, Christian Kral. *The New EDrives Library: A Modular Tool for Engineering of Electric Drives*. Modelica 2014.
- Christian Kral, Anton Haumer, Reinhard Wöhrnschimmel. *Extension of the FundamentalWave Library towards Multi Phase Electric Machine Models*. Modelica 2014.
- Christian Kral, Anton Haumer. *New Multi Phase Quasi Static FundamentalWave Electric Machine Models for High Performance Simulations*. Modelica 2014(a).
- Germar Müller, Karl Vogt, Bernd Ponick. *Berechnung elektrischer Maschinen*. Wiley, 2008.
- E. zur Nieden. *Berechnung von Stäben für Stromverdrängungsmotoren*. *Elektrotechnik und Maschinenbau* 51 (11): 129-134, 1931.

Simulation of Piping 3D Designs Powered by Modelica

Xavier Rémond¹ Thierry Gengler¹ Christophe Chapuis¹

¹Dassault Systèmes, Vélizy Villacoublay, France,
{Xavier.Remond, Thierry.Gengler, Christophe.Chapuis}@3ds.com

Abstract

Traditionally, piping systems have been defined in Modelica by connecting components in a model diagram. Additionally, the systems engineer must enter values for parameters such as pipes diameter and length, volume of vessels, etc. Those values are often also defined in CAD piping 3D designs, for example in CATIA by Dassault Systèmes. A more convenient definition of the piping system can be made directly using the data from the CAD environment.

A tool has been developed to extract data from CATIA piping 3D designs. This information is used to generate the corresponding Modelica representation.

Methodology based on the use of Modelica extends (inheritance) is applied to add controllers and other features to the generated model for dynamic simulation. Simulation results can be visualized directly in the 3D view of the piping design.

Specialized tools are developed, based on generated Modelica models, in order to enable quick calculations from the piping 3D design directly in the CAD environment.

Keywords: CAD, 3D, Piping, simulation, Modelica code generation

1 Introduction

Traditionally, piping systems have been defined in Modelica by connecting components in a model diagram and setting parameters value for those components. The components may be pipes with diameter and length, vessels with volume, etc. The piping network is partially duplicated in CAD piping 3D designs, which contains also piping components with parameters. More precisely, the piping 3D designs are master models for pipes shape as well as some other component parameters defined in the BOM (bill of material).

CAD designers and system engineers then need to exchange data in order to perform simulations based on actual piping 3D design.

A more convenient definition of the piping system can be made by directly using the data from the CAD environment. For that purpose, a tool has been developed to generate automatically a Modelica representation of a piping 3D design. It will help to perform various kinds of studies and simulations, by

simplifying tasks of users who are extracting data from piping 3D designs.

This article is structured as followed:

- Section 2 presents the type of models that are referred to in this paper;
- Section 3 illustrates the typical process involving the CAD designers and systems engineers;
- Section 4 explains how the Modelica representation of piping 3D designs is built;
- Section 5 focuses on the possible usages of Modelica representation of piping 3D designs.

2 Piping 3D models and Modelica models

In this section, models for piping 3D design and corresponding Modelica simulation models are defined.

2.1 Models for piping 3D design

Piping 3D designs are typically made of piping equipments and piping lines.

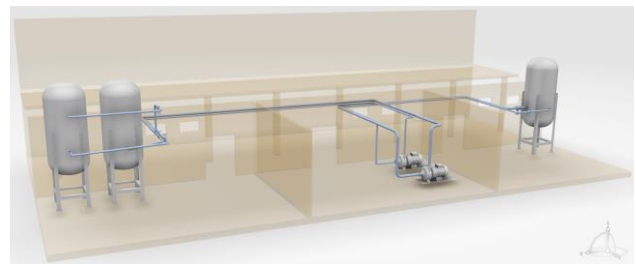


Figure 1. Example of piping 3D design

Equipments typically represent tanks, pumps, etc. In a CAD environment, they are parameterized 3D shapes.

Piping lines are composed of piping parts: pipes, valves, reducer, instruments, elbows, etc. Pipes may be either straight or curved.



Figure 2. Types of piping parts in CATIA

In a CAD environment, pipes are defined by a geometry (center curve + section), attributes (nominal size, end style, etc.) and a material. Other piping parts are parameterized 3D shapes.

Models of piping parts and equipment include connectors, also called ports. They dictate how part connect each other, and how a pipe is routed from the part. Each port has a 3D position and a specific attribute that dictates the type of a part that can be connected to it and the way it can be oriented. The following attributes are assigned to a port: port type (piping, electrical, etc.), nominal size, rating, end style, standard, outside diameter, wall thickness, alignment rules, orientation rules.

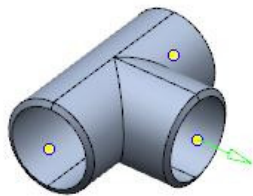


Figure 3. Typical piping "Tee" part with connectors

The 3D position of the connectors in a pipe model can be used in order to compute the altitude difference between the extremities of the pipe.

Additional data is available from the attributes of the piping parts, such as length and diameter for a pipe.

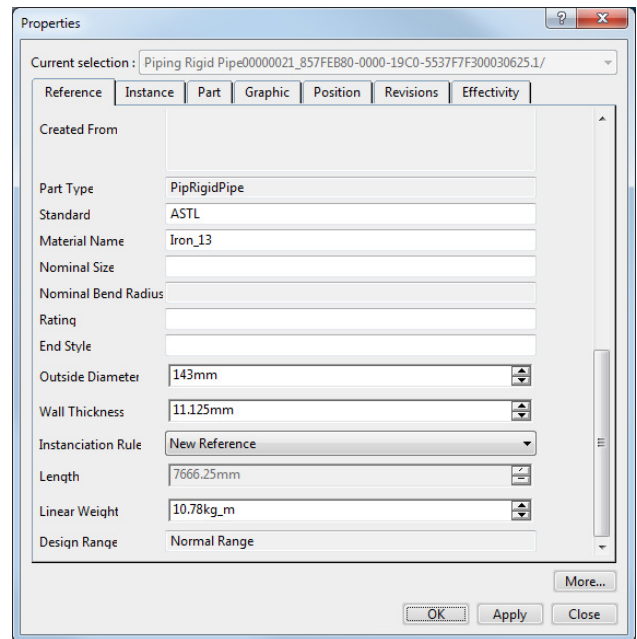


Figure 4. Attributes of a pipe in CATIA

Remark: It is possible to define flexible pipes in some CAD environments such as CATIA. However, such models are considered as rigid pipes in this project, mostly because of lack of availability for corresponding behavior description based on Modelica models.

2.1.1 Energy loss in piping parts

A specific attribute called Loss Coefficient K can be set on the piping parts (except for the pipes).

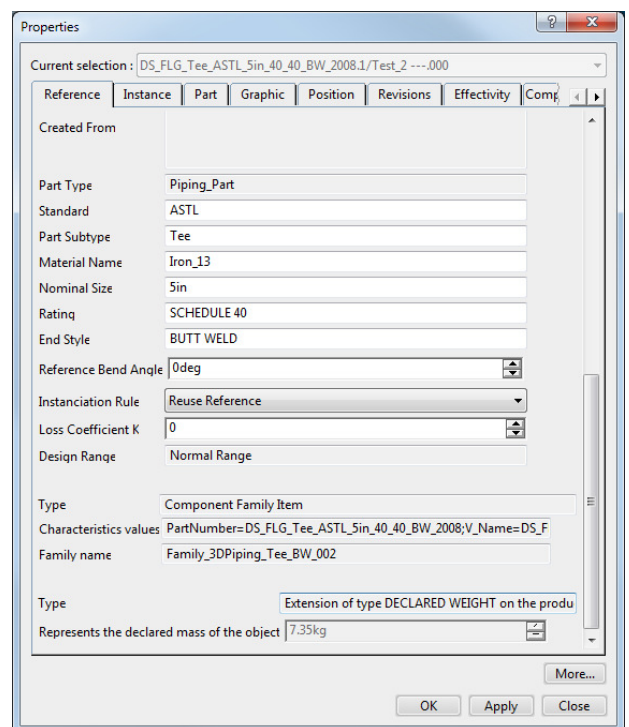


Figure 5. Attributes of Tee piping part in CATIA

It is based on Hydraulic Resistance Theory for modeling losses (Idelchik I.E., 1994). On a part, it is possible to compute the pressure drop Δp with a quadratic expression:

$$\Delta p = K \frac{\rho v^2}{2}$$

where ρ is the upstream density and v the mean velocity.

In Modelica the loss coefficient can be computed using the proper correlation as it is in *Modelica.Fluid* for the pipes, fittings or valves. For example in pipes, the Moody chart (Figure 6) is used to compute the loss factor due to the friction on specific circumstances (Casella *et al.*, 2006).

However, most piping parts do not allow a specific correlation model. The pressure loss coefficients are most often derived from measurements and are available in the form of charts. With the K attribute the correct coefficient can be integrated in the 3D part definition from manufacturer datasheet.

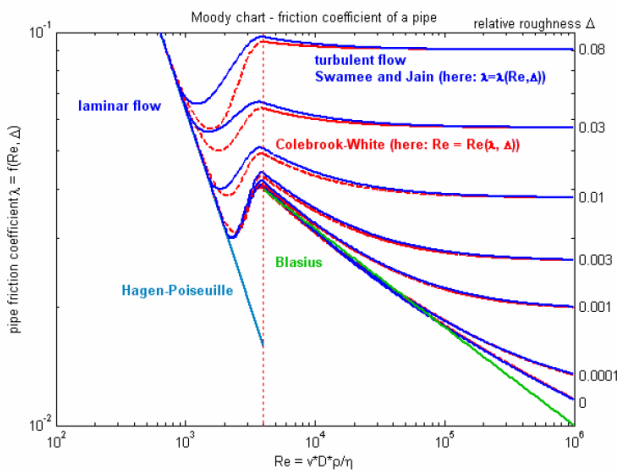


Figure 6. Moody Chart: $\lg(\lambda) = f(\lg(Re), \Delta)$, $\zeta = \lambda L/D$

2.2 Simulation models for piping 3D design

Variety of studies can be performed from piping 3D designs, depending on the kind of model and its purpose: hydraulic, pneumatic, thermo-fluidic, etc. As a consequence, multiple Modelica libraries can be used, separately or combined, in order to create a Modelica representation of those piping designs. For example, the following libraries may be used (<https://www.modelica.org/ModelicaLibrariesOverview#fluid>):

- *Modelica.Fluid*, from Modelica Association, for thermo-fluid flow in piping networks. *Modelica.Fluid* includes a subset of *FluidDissipation* library, from XRG Simulation. *FluidDissipation* contains heat transfer and

pressure loss calculations for industrial components used for the modeling of thermo-hydraulic processes.

- *Modelica.Media*, from Modelica Association, providing models and functions to compute media properties
- *ThermoPower*, from Politecnico di Milano, to model the dynamics of thermal power plants
- *AirConditioning*, from Modelon AB, to model transient and steady state behavior of air conditioning systems
- *Hydraulics*, from Modelon AB, to model hydraulic systems
- *Pneumatics*, from Modelon AB, to model pneumatic systems

Other libraries contain packages related to fluid, such as:

- *HumanComfort*, from XRG Simulation, providing models to estimate the human comfort within an air-conditioned zone
- *Buildings*, from Lawrence Berkeley National Laboratory, is a free open-source library with dynamic simulation models for buildings energy and control systems. It includes a package of models for pressure driven mass flow rate and for heat and moisture exchange in fluid flow networks.

This list is of course not exhaustive.

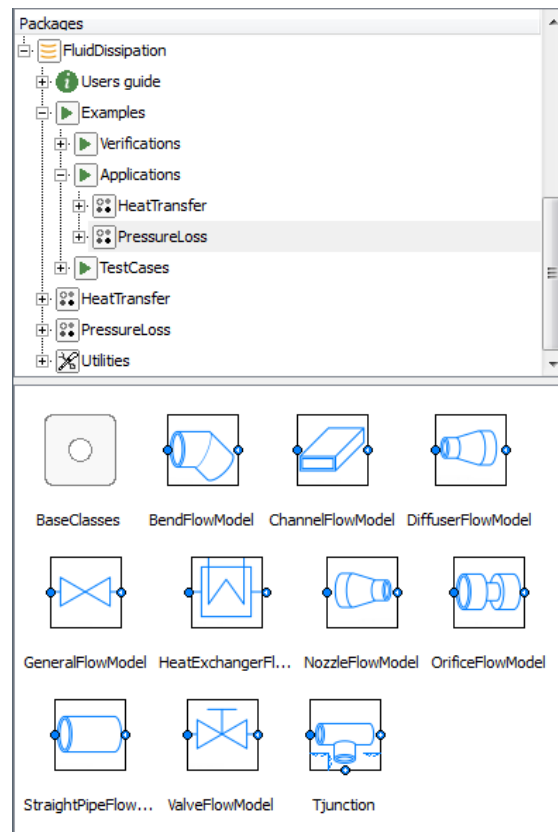


Figure 7. FluidDissipation library

Depending on the Modelica libraries used, the necessary data from the piping 3D design can vary. The geometric data are the most commonly used: length and diameter of pipes, bend radius, volumes, etc.

Other data, such as material or roughness, may be needed in the Modelica representation of the piping design.

3 Collaboration between CAD designers and system engineers

As described in the previous section, CAD designers and system engineers share a large amount of data and even more if the cycle of development is considered.

3.1 Parameter extraction and matching

On the upper picture of Figure 8 a pipe 3D geometry is presented in light grey color. The interesting point on this example is the number of parameters necessary for a Modelica based system simulation. Considering the diameter, the bend radius, the angles of curvature and the height variations, this geometry contains at least ten parameters. This is already significant for a single pipe.

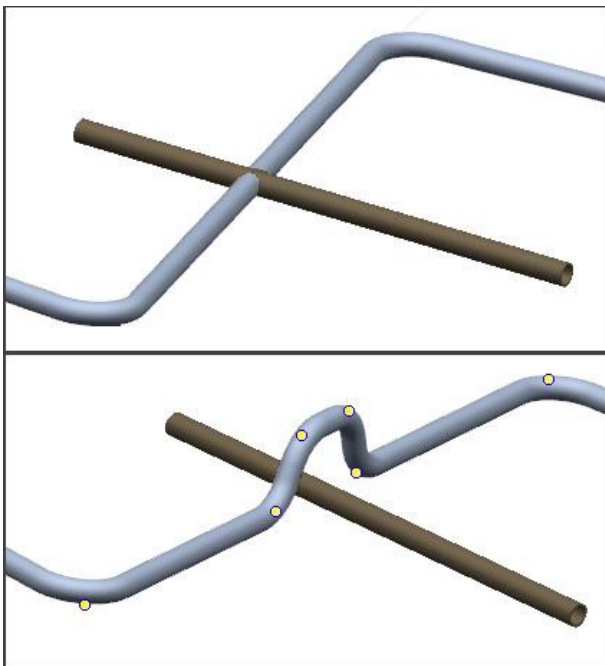


Figure 8. Typical modification of piping 3D design

In reality, complete 3D models are much more complex and contain multiple piping parts (valves, reducer, junction...). Consequently the number of parameters can increase, up to hundreds or even more.

It is therefore useful to define an automatic extraction and a matching method from each 3D parameter to the system simulation model.

3.2 Life cycle of models

When considering the life cycle of the models, the number of parameters to be shared between piping 3D designs and piping simulation models becomes even more critical. For each design change, as shown on Figure 8, the Modelica model must be updated accordingly, including the parameters. A system simulation model in this context strongly depends on changes in the design and more if the design is (partially) driven by the results of simulation. The number of parameters is then even more difficult to manage due to the time that it consumes and also the increased risk of error.

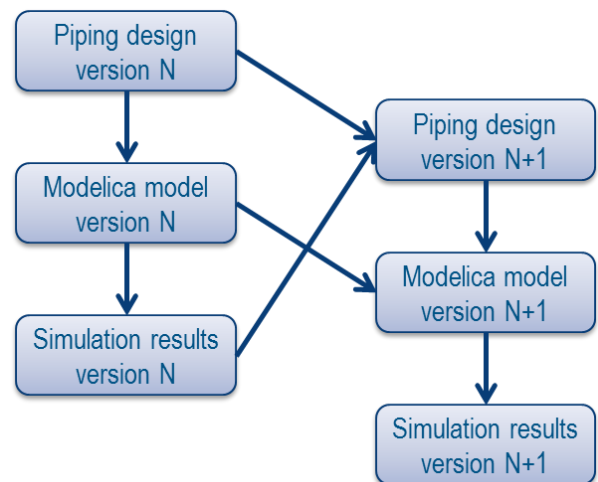


Figure 9. Life cycle of models and data flow without generation tool

Consequently, it is necessary to take care of the life cycle regarding the Piping Design, the Modelica model and also the simulation results. On Figure 9 the usual cycle is presented. With the Modelica generation tool presented in this paper, this cycle can be simplified (Figure 10). The transition from piping design to Modelica model is now partially automated and after each design modification a new model can be generated automatically. Moreover, data manually added by user in Modelica model is automatically reused with the new version of generated model. In that process, the piping 3D model is the master model which hosts the piping system data.

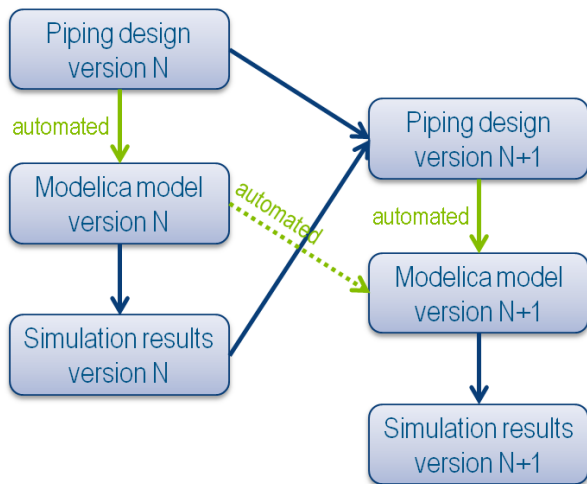


Figure 10. Life cycle of models and data flow when using the Modelica generation tool

4 Modelica code generation

The Modelica code generation from piping 3D design is done in a few steps.

4.1 Traversing piping 3D designs

The first step is done by traversing and filtering the 3D design. The algorithm collects all the information available in the parametric geometry and in the model based on a specific semantic. The 3D design is then mapped and may be assimilated to a graph with each nodes corresponding to a pipe, a piping part or equipment.

The approach is similar to a previous work with mechanical models (Elmqvist *et al*, 2009).

4.2 Generated Modelica code

From the graph built by traversing the piping 3D design, Modelica code is generated. A root Modelica model is created. Each node in the graph is represented by a Modelica component in this root model. The reference class of each of these components will be determined by the mapping process. Every component is replaceable; this gives more flexibility if some components must be changed afterwards.

Each node of the graph contains the type of its corresponding piping object. That type is mapped with a Modelica class or a group of classes by a mapping

table. Depending on the Modelica libraries that are targeted for the simulation, that table can be modified in order to map piping types with the Modelica classes which are of interest.

It is important to note that the mapping of a single 3D design object can involve several basic Modelica classes such as shown on Figure 11. The rigid pipe is a single entity in term of 3D design. However for the simulation this pipe needs to be split in order to take in to account the bends for example. In any case and to simplify the generated root model each 3D design object which needs several Modelica base classes is encapsulated in a sub-class.

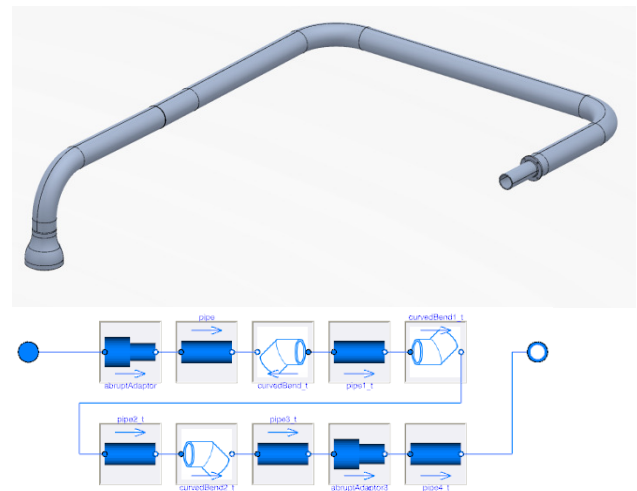


Figure 11. 3D pipe and its Modelica representation

To complete this approach the mapping process also includes the mapping of the attributes. The program exposes keywords associated to a piping type. The user is then responsible of the matching definition with the input parameter for each of the chosen classes (Table 1). In the current prototype the mapping table is defined from a CSV text file. For example, the parameter of Modelica class can be mapped to a formula including the K attribute introduced in section 2.2.1.

The next step will consist in developing convenient editor for the mapping.

Type	Keyword from 3D	Modelica Path	Modelica Parameter
PipRigidPipe		Modelica.Fluid.Pipes.StaticPipe	
	Port_1		port_a
	Port_2		port_b
	Diameter		diameter
	Length		length
	Height		height_ab

Table 1. Example of mapping table

4.3 Diagram layout generation

The understanding of the diagram is a non-negligible point for Modelica model and even more in the case of automatic generation. Currently a simple force-based algorithm provides an understandable layout. Figure 12 is the resulting layout when generating the model of Figure 1.

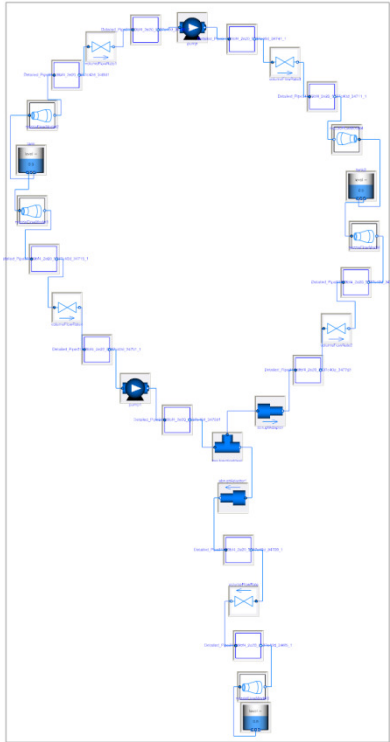


Figure 12. Diagram generated from model of Figure 1.

The implemented layout algorithm is good at displaying the loops and symmetries on the diagram, as shown in Figure 13. It is also fairly quick: the number of components in a Modelica diagrams is relatively small compared to some other kind of graphs, and the algorithm computes the layout almost instantaneously for diagrams containing less than 50 components.

However, the current algorithm has some limitations including the following:

- The algorithm sometime converges to a local minimum of energy instead of a global minimum, which results into lower quality layout. This is the most common disadvantage of force-based layout algorithms.
- Position of connectors in the component icon is not taken into account; as a consequence, even if the components are nicely positioned, the routes for connections may be not so good.
- Computed layout is not optimized for diagrams with Manhattanized connections, as the components are not aligned;
- Some generated diagrams are not compact, as shown in Figure 12 and Figure 16.

Force-based layout algorithms are quite flexible. They can be extended or combined with other algorithms to improve the generated diagrams. Such improvement may be part of a future work.

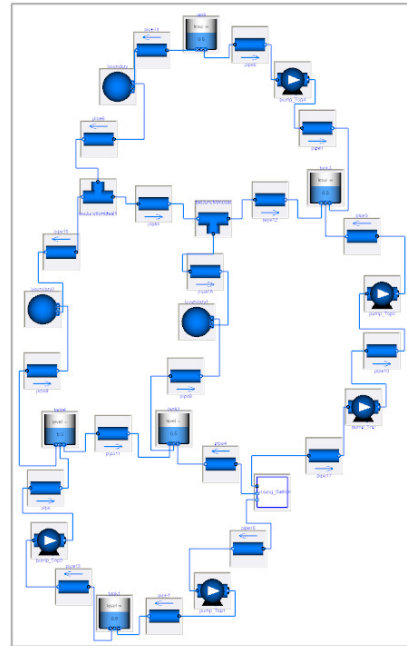


Figure 13 Example of diagram generated with force-based layout

In the context of piping design, a 2D schematic design is often available. This could provide a strong base to generate a Modelica diagram with a nice layout for people who are used to work with P&ID schematics. In future works, generation of Modelica model based on both Schematics and 3D CAD will be investigated.

5 Applicative usage of generated models

5.1 Model completion

An important property of the generated model is the possibility to use Modelica extends (inheritance) for adding controllers and other features to the model for simulation and also for setting missing parameters values. It is also possible to replace components if needed, because all components are replaceable in the generated model. In that way, the generated model is separated and can be changed independently of the added and/or modified components.

In order to insert inline components in the generated model, for example inline sensors, it is not possible to change the connections in the extended model. The following methodology can be applied instead. A new class can be created which wraps the generated class for the pipe which includes the new inline component. Then, in the extended model, the initial component is replaced with the new class (Figure 14).

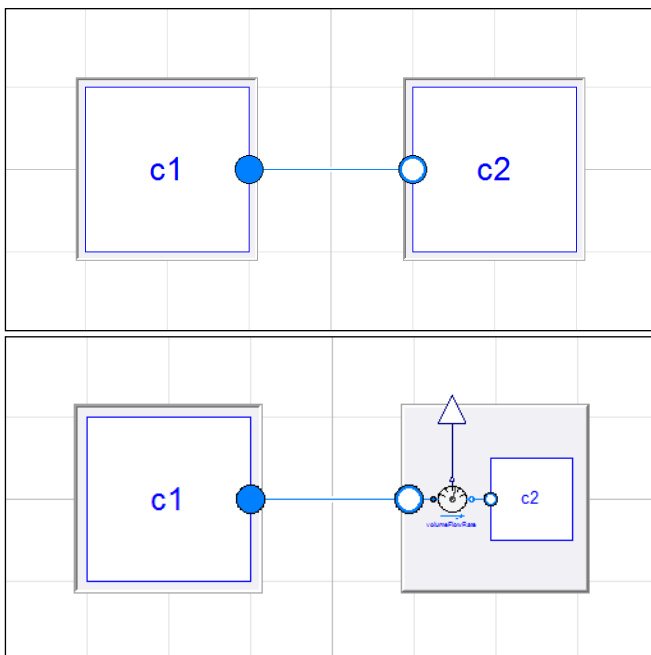


Figure 14. Initial generated model and extended model with inline sensor

A main limitation when working with extended models is related to diagram layout, which cannot be modified.

5.2 Model update

Thanks to the use of the extended model, the modifications on the Modelica representation performed by the user are not lost when the Modelica representation is updated.

Even with the recommended methodology to have seamless update of the Modelica representation, it is useful to have a comparison tool so the user can easily identify the changes resulting from the update operation. Highlighting those changes helps validate the changes and identify where the new Modelica representation would need some additional information or fine tuning.

5.3 Model simulation

The prototype of the Modelica generator has been used to compare alternatives of 3D routes for pipe lines.

The system considered here is a simple civil engineering system for water supply. It consists of 6 main installations distributed over an area of 2km by 2km:

1. Intake pumping station
2. Intermediate pumping station
3. Water treatment and storage plant
4. Industrial plant
5. Industrial plant
6. Business park

As presented on Figure 15 a piping network is defined in order to distribute the water to each plant based on its need. The apparent simplicity of this design actually conceals already significant amount of parameters (length, diameter, height variation, roughness...) that must be taken into account for a precise definition of the system, even in a pre-phase project.

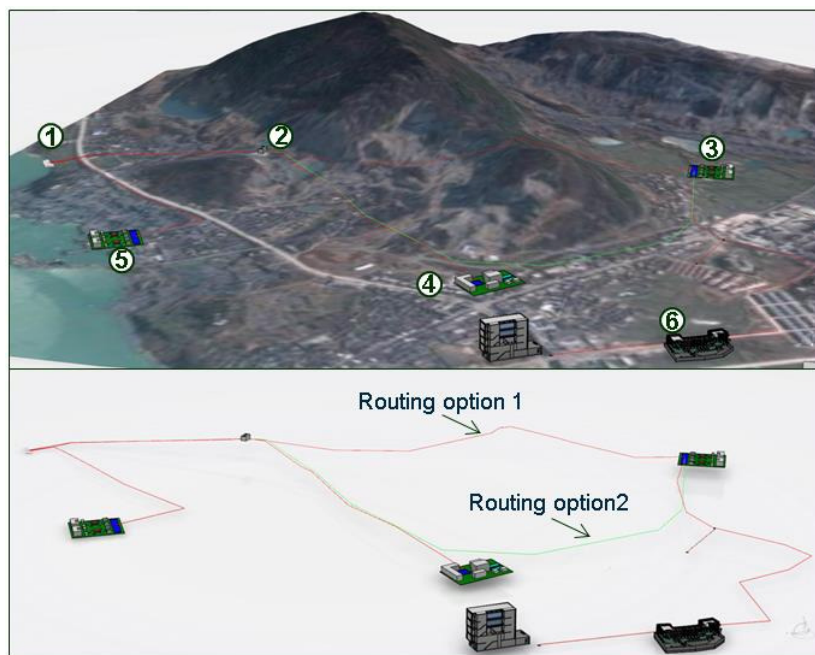


Figure 15. The water supply piping 3D design. At upper part: design with environment. At lower part: without environment

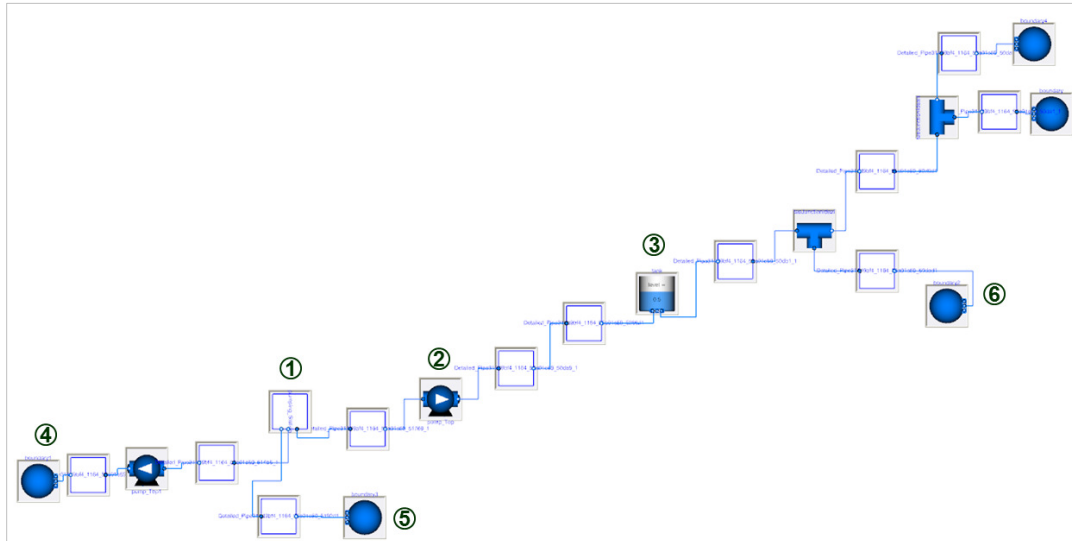


Figure 16. Modelica Diagram generated from the water supply piping 3D design

The study we propose focuses on the design of the branch between the Intermediate pumping station (2) and the Water treatment and storage plant (3). Two routing options are proposed (lower part of Figure 15). The option 1 is passing over the mountain and the option 2 gets around it with a pipe additional length of 750m.

By using the Modelica representation generator it is possible to create a simulation model where we find each installation and the network structure. Figure 16 shows the generated Modelica diagram according to a specific mapping with the Modelica Standard Library, Buildings Library and also with our own test library for the pumping station behavior definition.

Once the mapping is defined by the user only few seconds are needed to generate a Modelica representation from the piping 3D design.

From the generated models, simulations have been performed for each 3D design. It is good to note that to perform this simulation a complementary work is necessary in Modelica. For example, the initial conditions need to be precisely specified in order to have a functional simulation; therefore an expertise in simulation is required for the first model initialization.

The first result that we can extract from the simulations is a pressure drop characteristics comparison between the two options. On Figure 17, route 1 (passing over the mountain) has a lower pressure drop for any volumetric flow rate. This result seems to demonstrate that the friction losses on the longer route 2 have an important impact on the pressure drop. The pumping system will logically consume more power with routing option 2.

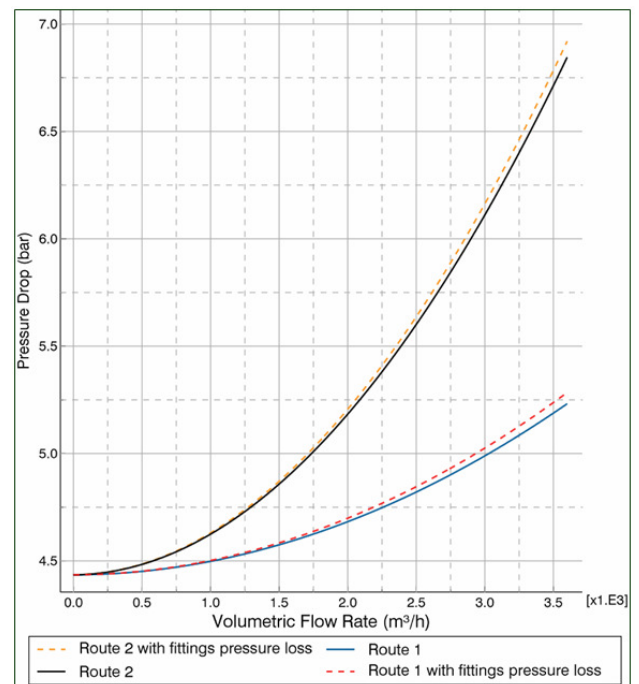


Figure 17. Pressure drop characteristics for the two routing options with incompressible water

However, even if we can already estimate the additional power needed by the second option, it is easily possible to confirm this with a simulation which will take into account the full system. After results analysis, it appears that for a volumetric flow rate of 2500 m³/h, route 2 will consume more than route 1, from 330kW to 369kW at the intermediate pumping station.

These results can then be the starting point to trade-off discussions in order to choose the best design, concerning: the cost to build the system, the choice of material or also the maintenance costs. In any case, these conclusions are based on a simulation which is

strongly linked with a 3D design that could be quickly updated and tested. This example illustrates the possibility to improve the design process with system simulation.

5.4 Analysis tools powered by Modelica

In addition to the global approach of complex fluidic network for simulation, specialized tools are developed. Based on generative Modelica model, they will allow quick calculations from the piping 3D design directly in the CAD environment.

For example, a command computes pressure drop in a single rigid pipe. The user will have only to enter few parameters, and the tool will provide the results of the computation. The underlying Modelica model does not need to be shown to the user. This system engineering calculator aims at performing quick pre-dimensioning of the designs, before performing more detailed simulations.

This tool is currently under development.

5.5 Animation of piping 3D design

As the piping 3D design is the original model for the simulation, it is natural to benefit from that design in order to convey information, such as simulation results.

Fluid flow in pipes is typically a piece of information that can be displayed on the 3D mockup, based on animation.



Figure 18. Animated flow on 3D design

A new dedicated Modelica class has been developed, which works like a sensor in order to collect data from the piping model. That data is used by an animation module in the 3D view. Animation is performed by moving a texture along the pipe.

Compared to the existing class Modelica.Mechanics.MultiBody.Visualizers.PipeWith

ScalarField, the new class has the advantage to work with bended pipes. Also it is a light-weight class, as it is designed for a more specialized purpose of flow animation. As a result, it has a tiny impact on the size of simulation results. Such components could be standardized, as the pipe shape data is fully defined in Modelica component.

This class has two main sets of attributes, one for the description of the geometry and one for the description of the texture.

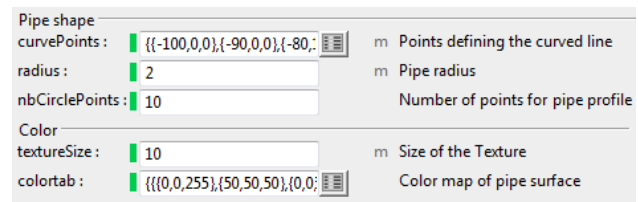


Figure 19. Modelica parameters for the pipe flow animation

The geometry is described with a set of points defining the spine of the pipe, the radius of the pipe, and the number of points along the circle defining the pipe profile.

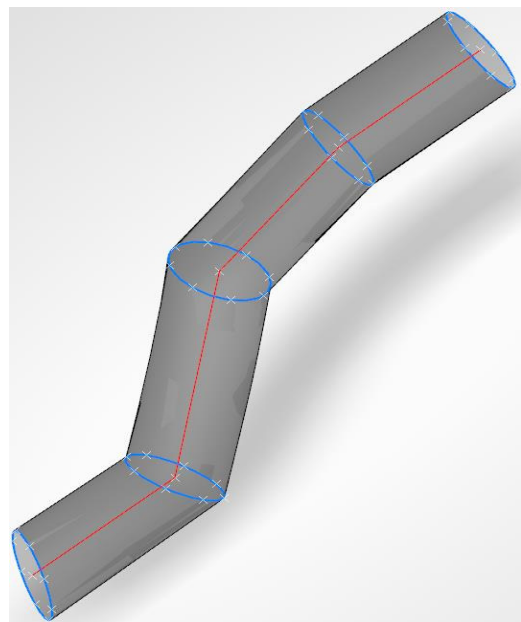


Figure 20. Illustration of pipe geometry with defining points highlighted

The texture is described with an array of RGB color values stored in an array of size U x V, where U and V are the number of colors respectively along the pipe profile and along the pipe axis.

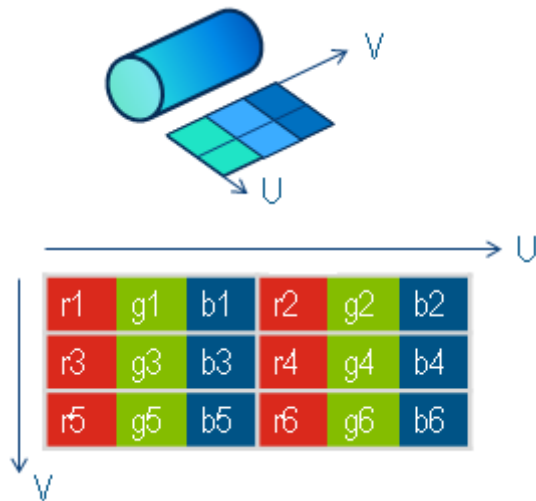


Figure 21. Color encoding in the color map

Let us consider the following simple model:

```

model TestAnimation
  CurvedPipeAnimation pipeAnimation(
    nbCirclePoints=10,
    curvePoints= {{0,0,0},{10,0,0},
                  {20,10,5},{30,15,5},{40,15,5}},
    radius=3,
    colortab = {{0,90,255},{10,120,255},
                {10,150,255},{10,120,255}}},
    textureSize=10,
    texturePosition=time);
end TestAnimation;

```

Figure 22. Example of model using a curved pipe animation

During the simulation, the following pipe animation will be generated:

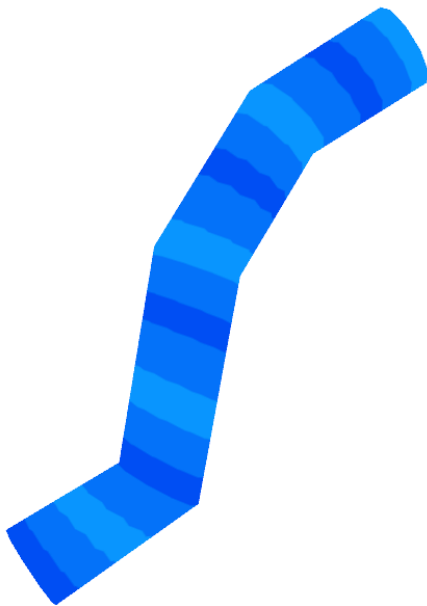


Figure 23. Curved pipe animation example

The animation is based on one scalar input, called `texturePosition`, corresponding to the position of the texture along the pipe. In generated Modelica representations of piping 3D design, an equation links that input with the flow rate.

6 Conclusions

With the proposed approach of generative Modelica representation of piping 3D designs, the collaboration between CAD designers and system engineers will become easier. The automated exchange of data improves efficiency and reduces the risk of errors. The editable mapping table offers high flexibility for multiple usages of the Modelica code generator.

Moreover, it will bridge the gap between the CAD and System Engineering disciplines, by providing easy-to-use utilities for pre-dimensioning of systems directly available in CAD environment.

Future work will include the development of such tools for CAD users, of editor for the mapping table, and improvement of diagram layout.

Acknowledgements

The authors would like to thank Guillaume Lerey, Gustavo Passini, Simon Royer (Dassault Systèmes SE), Markus Andres (3DS GmbH) and Dr Hilding Elmqvist (Dassault Systèmes AB) for their contributions and feedbacks.

References

- Casella F., Otter M., Proelss K., Richter C., Tummescheit H. The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. *Proceedings 5th Modelica Conference*, pp 631-640. 2006.
- Elmqvist, H., Mattsson, S. E., & Chapuis, C. Redundancies in Multibody Systems and Automatic Coupling of CATIA and Modelica. *Proceedings 7th Modelica Conference*, pp 551-560. 2009.
- Idelchik I.E. Handbook of Hydraulic Resistance. 3rd edition, *Begell House*, 1994. ISBN 0-8493-9908-4.
- Vahlenkamp T., Wischhusen S. FluidDissipation for Applications - A Library for Modelling of Heat Transfer and Pressure Loss in Energy Systems. *Proceedings 7th Modelica Conference*, pp 132-141. 2009.

3D Schematics of Modelica Models and Gamification

Hilding Elmqvist¹, Alexander D. Baldwin^{1,2}, Simon Dahlberg²

¹Dassault Systèmes, Lund, Sweden, Hilding.Elmqvist@3ds.com

²Malmö University, Malmö, Sweden, {alexander.d.baldwin, simondahlberg89}@gmail.com

Abstract

Block diagrams have been used for a long time to express data flow in dynamic models, i.e. the input output relations between calculation blocks. SysML diagrams are also used to express other relations such as component hierarchy and inheritance. Modelica uses object diagrams, a generalization of block diagrams since acausal connections are allowed. CAD uses a 3D representation to represent the assembly of a mechanism, i.e. how bodies are coupled with joints. This paper describes a generalization of object diagrams, called *3D Schematics*, to utilize 3D representations of the icons/shapes and unification with assembly diagrams and exploded views.

The ideas have been prototyped in a program called Playmola which is inspired by computer games. The goal is to make a model authoring environment that is much more intuitive and fun than existing ones. The hope is that such a tool would be used to promote science for students already in high-school.

Keywords: Block Diagrams, Object Diagrams, MultiBody Assembly, Exploded View, Gamification

1 Introduction

The look and feel of Modelica tools needs to be modernized. It has roots from the 1990s when compromises due to limitations in rendering speed needed to be made with regards to capabilities. As a consequence, a flat 2D graphics representation was introduced in Modelica.

Watching young children experience, handle and enjoy 3D scenes and actions in Minecraft¹ inspired making a completely different 3D environment for building and experiencing Modelica models. The 2D physics environment Algodoo (Algodoo, 2015) has also served as inspiration.

2 Modelica Object Diagrams and 3D Schematics

3D schematics as a unification of visual representations of models will be presented by a series of examples.

2.1 MultiBody Modeling

An example of how to model MultiBody systems in Modelica is shown in Figure 1. It is a pendulum called the Furuta pendulum.

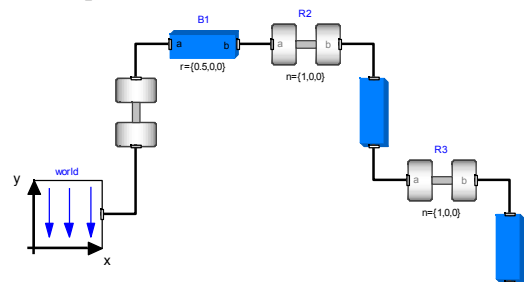


Figure 1. Furuta pendulum model in Dymola

The corresponding 3D schematic of Playmola is shown in Figure 2.

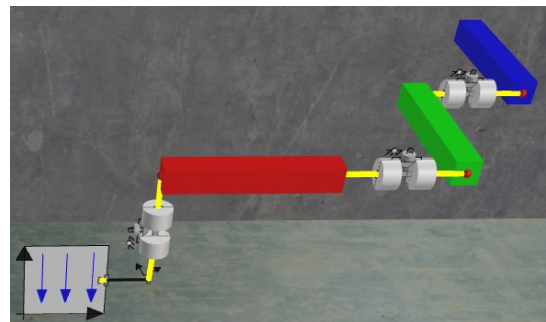


Figure 2. Furuta pendulum model in Playmola

Since it's a 3D view the user can change the camera position to get a different perspective of the Furuta pendulum. It should be noted that the joints are rendered in 3D and are oriented according to the axis of motion. Red spheres represent the connectors.

The 3D representation of the revolute joint has been automatically derived from the Modelica annotation of the icon which contains:

```
Rectangle(  
  extent={{-100,-60},{-30,60}},  
  lineColor={64,64,64},  
  fillPattern=FillPattern.HorizontalCylinder,  
  fillColor={255,255,255},  
  radius=10),
```

¹ <https://minecraft.net/>

Since the fillPattern is set to HorizontalCylinder, a cylinder is generated for the 3D representation. The standard 2D and the “2.5D” representations are shown in Figure 3.

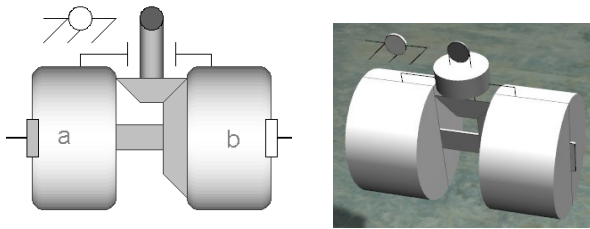


Figure 3. Standard 2D and new “2.5D” visualization of the revolute joint

The image for the poster of the Modelica Conference 2015 in Versailles, Figure 4, was generated based on a gearbox model using this “2.5D” technique.

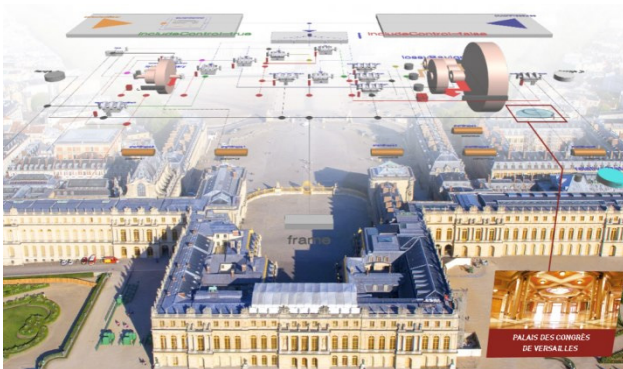


Figure 4. Modelica Conference poster using “2.5D” visualization of a gearbox

The view of the Furuta pendulum in Figure 2 can be seen as an exploded view of a 3D assembly drawing. The unexploded view is shown in Figure 5. The joints and connections are then not rendered.

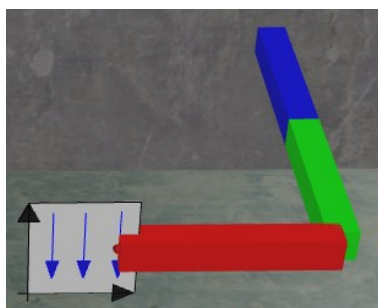


Figure 5. Un-exploded view of Furuta pendulum

The above view is also the animation view for Playmola. It corresponds to the special animation view of Dymola shown in Figure 6.

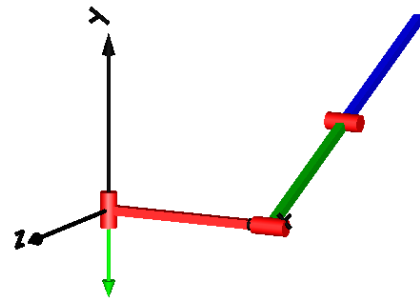


Figure 6. Animation of Furuta pendulum model in Dymola

2.2 CAD shapes – Robot Model

The mechanics part of the model: Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot is shown in Figure 7.

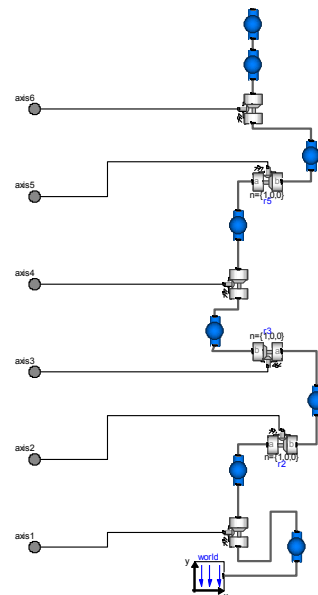


Figure 7. Robot model in Dymola

The corresponding non-exploded Playmola model is shown in Figure 8.

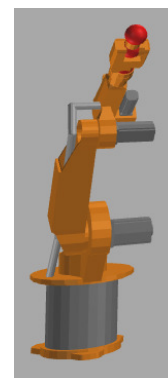


Figure 8. Un-exploded view of Robot model in Playmola

This view uses the actual shapes of the bodies instead of icons and is constructed in the same way as the assembly is made.

Figure 9 shows the exploded view.

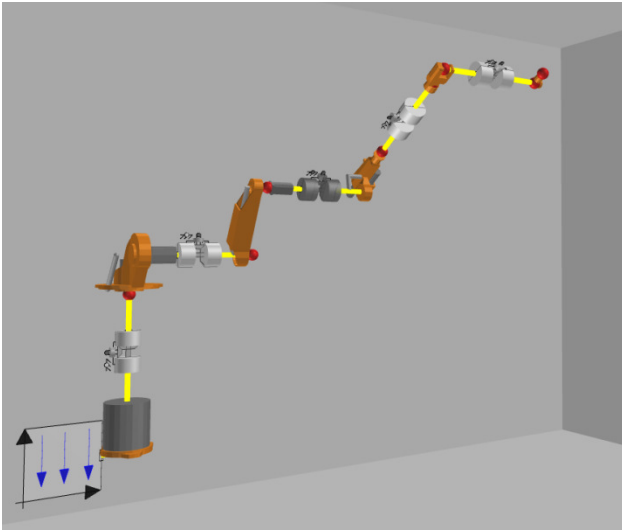


Figure 9. Exploded view of Robot model in Playmola

This is a very intuitive visual representation of the robot in which the degrees of freedom are clearly shown. In this representation, additional modeling elements can be introduced such as electrical motors and gearboxes as shown in Figure 10 of the partial robot model (the 3D representations of the motors and gearboxes were automatically generated):

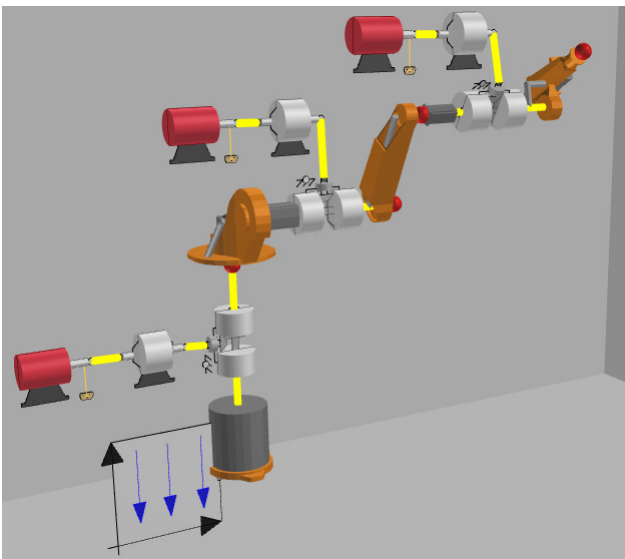


Figure 10. Exploded view of Robot model with motors and gearboxes in Playmola

2.3 Bodies in Contact

Many real life situations involve bodies in contact or colliding bodies. Playmola utilizes the new functionality available in Dymola for contact handling (Elmqvist, et al., 2015). It is therefore possible to put some boxes on a table (big box), set initial rotation speed on one of the boxes, and experience the domino effect as shown in Figure 11.

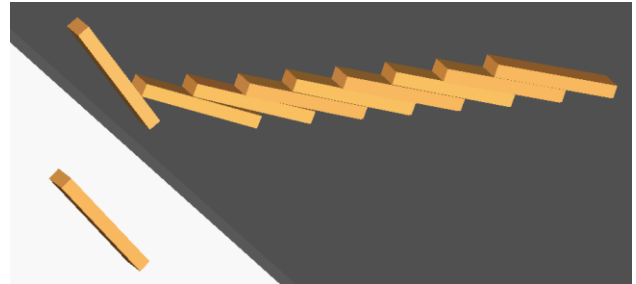


Figure 11. Domino bricks model in Playmola

3 Gamification

3.1 Background

Science education has been slow to adopt the use of interactive simulations. (Wieman and Perkins, 2006) are vocal proponents of using interactive simulations as an educational tool in science and argue that traditional forms of education fail to provide students with an understanding of science by suppressing their interest in the subject and failing to engage them. Wieman and Perkins argue that interactive simulations are an effective complement to traditional media in science education – an idea that is supported by research into the use of simulation tools to engage college students (Podolefsky, 2010). According to Wieman and Perkins, the most important features of an engaging interactive simulation tool are:

- Highly interactive animation
- An appealing environment and sophisticated graphics
- Simple and intuitive controls
- Connections to real-life objects (Wieman and Perkins, 2006 p. 291)

Research shows that an effective way of promoting user-engagement and interest in learning environments is the incorporation of elements from digital games (Sabourin and Lester, 2014). The use of game-elements in other contexts, usually referred to as gamification, has been an increasingly popular research topic in recent years, with many studies applying its principles in an educational context (Hamari et al., 2014), (Seaborn and Fels, 2015). The stated motivation for the use of (digital) gamification is often to increase user

engagement and, consequently, user retention in software systems (Deterding et al., 2011).

Most gamification today is implemented using extrinsically motivating elements such as points, achievements and leaderboards - encouraging users to compete for rewards and status. This kind of gamification has been criticised for not being representative of what makes games fun and has been referred to as “pointsification” (Robertson, 2010). Numerous attempts have, however, been made to construct models or frameworks for gamification which focus on intrinsic motivation and fun.

In addition to design-elements from games, game-related technologies such as game engines and 3D-engines can be useful in non-game contexts and have successfully been used in the development of simulation tools on numerous occasions (Bijl et al., 2011). Game engines often boast advanced graphical features with realistic lighting and shadows and support for importing complex animated and textured 3D models, which can be used to provide the appealing environment and sophisticated graphics Wieman and Perkins consider important. Depending on the type of simulation tool to be created, other common features of game engines such as built-in physics engines and artificial intelligence systems can also be useful. These types of features are often provided in a development environment which supports rapid production, commonly with little need for programming.

3.2 Gamification Frameworks

(Nicholson, 2012) attempts to solve the issue of the negative effects of extrinsic motivators on intrinsic motivation by conceptualising a framework for “meaningful gamification” with a focus on the end-user rather than the organization providing the service, even providing his own definition of meaningful gamification as: “the integration of user-centered game design elements into non-game contexts”. Besides recommending a focus on intrinsic motivators, Nicholson’s framework stresses the importance of context in the use of game elements, referring to the concept of situated motivational affordances (Deterding, 2011), which describes how the motivational effect of a system element depends upon the background of the user and the context in which it is used in the system. Nicholson concludes that meaningful gamification relies on using elements that users with a wide variety of backgrounds can relate to in the right context within a system. In his examples of applications of meaningful gamification, Nicholson suggests removing scoring/rule-based elements and focusing on ‘play’, referring to this as ‘playification’. Deterding et al. refer to this concept as ‘ludification’ and consider gamification a subset of ludification (Deterding et al., 2011 p. 13).

Numerous studies on gamification refer to Malone’s paper (Malone, 1982) on using ideas from games in effective user interface design. While written long before the coining of the term “gamification”, Malone’s ideas still have a lot in common with the aforementioned definitions, particularly Nicholson’s framework for user-centered meaningful gamification. Malone focuses on the user’s enjoyment, defining three principle heuristics for designing enjoyable user interfaces: challenge – the activity should have a clear goal and an uncertain outcome, fantasy – the interface should be “emotionally appealing” and use metaphors that the user can relate to, and curiosity – the interface should provide the right level of informational complexity in order to be novel or surprising while still understandable; it might also use “sensory curiosity”, which refers to the use of audio and visual effects as decorations, to enhance fantasy or as a means of representing aspects of a system.

3.3 Use of Gamification in Playmola

Playmola was constructed with gamification in mind, primarily taking advantage of Malone’s “fantasy” to immerse users in the setting of a workshop where they can experiment and build models in an environment that is both familiar to them (most people have some experience of building things in a garage or shed) and contextually appropriate for the kind of construction being done (mechanical models in the developed prototype), see Figure 12.

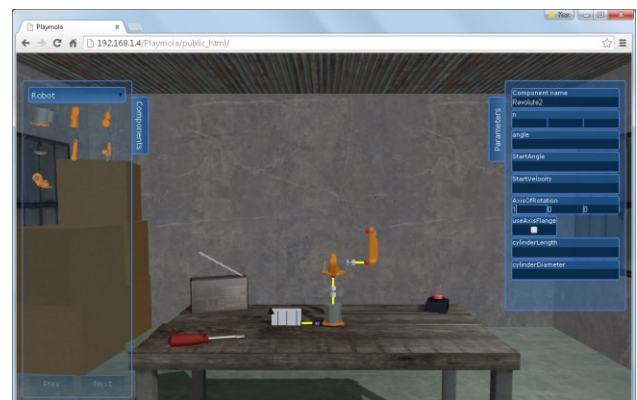


Figure 12. Garage environment for visual experiments in Playmola

Textured 3D models for the environment (including walls, floor, ceiling, a workbench, a screwdriver and stacked cardboard boxes), particle effects (a welding effect using sparks, see Figure 13) and music and sound effects (specially composed background music and a welding sound effect when components are connected) were all used to invoke this sense of fantasy.

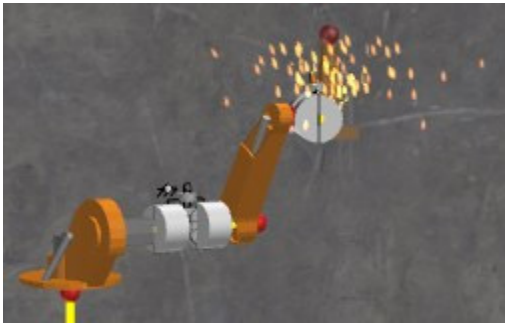


Figure 13. Welding a joint in Playmola

Playmola has to a certain extent been inspired by the Kerbal Space Program² which allows you to construct and launch a rocket built-in a hangar, see Figure 14.



Figure 14. Kerbal Space Program

4 Modeling with Playmola

As demonstrated above, Playmola allows modeling with the predefined 3D objects, Box, Cylinder and Sphere in addition to objects defined by triangular meshes imported from CAD programs. The components are organized in groups, see Figure 15.



Figure 15. Group of Robot parts

² <https://kerbalspaceprogram.com/>

When hovering over the group, the 3D parts are rotated in order to give better perception. It should be noted that the same 3D representation is used in composition mode as in animation mode. In addition, all Modelica library components can be used when modeling. Their visual representation is usually flat and only shown in the exploded view.

When a connector of a part is moved close to a connector of another part, the connectors are highlighted. A dialog is shown (Figure 16) to allow selection of joint type to be inserted.

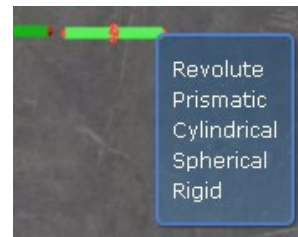


Figure 16. Dialog for selecting joints when connecting parts

When parts and joints are connected in a loop (kinematic loop), special handling in the Dymola solver is required. To simplify for the user, such a situation is automatically detected and currently a strong spring damper is inserted to act as a cut joint and brake the kinematic loop. This cut joint is visualized with a dashed line (Figure 17).

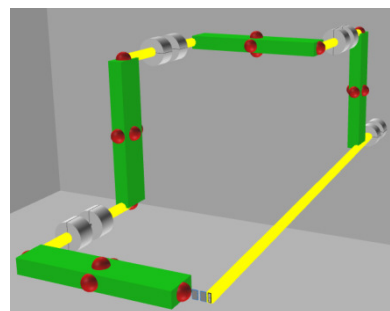


Figure 17. Kinematic loop

In order to promote ease of use for younger users, most components used in Playmola are simplified wrappers of components from the Modelica Standard Library, dramatically reducing the number of parameters the user can change to a subset of key parameters that allow the user freedom to experiment without becoming confused by an overload of information.

For example, when a revolute joint is selected a dialog slides in from the right (Figure 18) showing only certain parameters.



Figure 18. Simplified dialog for revolute joint

These kinds of constraints are recommended by (Podolefsky et al., 2010) in their discussion of the use of simulation tools in physics education: ‘constraining what students can do reduces cognitive demands and frees up resources for sense making and development of an expert-like mental framework.’ In addition to these simplified components, other components can be loaded as desired.

5 User Experiences

5.1 First Grader

Playmola was presented to a 7-year old boy. He is interested in machines and had previous experiences with simpler 2D educational games. The first exercise was to construct a double pendulum, i.e. introducing the concepts of setting the size of a box and coupling the parts by means of revolute joints. This was carried out without problems especially since changing a parameter such as length is immediately visualized.

The next exercise was to construct a robot using the set of bodies shown in Figure 15. Axis of motion then becomes very important. The concept of direction was introduced by showing 3 fingers pointing right, up and towards him and corresponding to the 3 input boxes in the dialog. It was explained that 0,0,1 meant choosing the axis towards him and 0,1,0 meant up. It was also explained how to move the parts by setting the StartAngle by a short introduction to the concept of degrees for angles. After that he could build the robot as shown in Figure 9.

5.2 Bachelor Students

In a small-scale study, eight users were about to recognize Playmola’s theme and felt that it was enjoyable and suitable for the application, using words like ‘charming’ and ‘great’. One user described how the environment actively enhanced the user experience: ‘You come into a calm environment with calming music which isn’t distracting. It makes it easier to concentrate on what you actually want to do.’

The use of a single, unified 3D view for construction and animation of models - instead of the disconnected 2D view for construction of models and 3D view for animation of simulation results seen in Dymola -

appears useful from the perspective of helping new users understand the application. All users agreed that the 3D environment made it easier for them to visualize the models, one pointing out that the unified view made it easy to tweak parameters in a model and get direct visual feedback reflecting the changes. Observations also showed that the ability to view the complete model in 3D allowed users to quickly discover the source of any issues.

6 Proposed Extensions to Modelica

As demonstrated above, Rectangle with fillPattern=HorizontalCylinder can be interpreted as a Cylinder enabling the “2.5D” representation. Other graphical primitives such as Ellipse, Rectangle and Polygon just get a fixed small extrusion in the z-direction. This means that, for example, the icon of the Prismatic joint does not look right, see Figure 19.

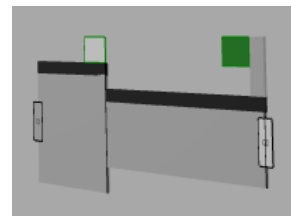


Figure 19. Standard prismatic joint rendered in Playmola

We propose that general 3D primitives are introduced in the graphics annotations of Modelica. In particular, a triangular mesh representation gives a very flexible way of defining the component shapes.

The paper (Elmqvist, et al., 2015) presents details about modeling with triangular meshes. A popular representation is an array of 3D vertices and an array of triangles defined as 3 indices into the vertices array. A set of functions are defined for building triangular meshes of Box, Cylinder, Sphere and other elementary shapes. Transformation functions for translating, rotating and scaling of shapes are defined. In addition, functions from extrusion of shapes from polygons are available. Finally the Constructive Solid Geometry (CSG) operations, union, difference and intersection are defined.

By replacing the four rectangles of the Prismatic joint icon graphics by the following annotation:

```
TriangleMesh(mesh=
  rotate(
    union(
      translate(
        Box(size={200, 60, 60}),
        r={-100, -30, 30}),
      translate(Box(size={70, 100, 100}), r={-100, -0, 50})),
    angles={3.14159265/6, 0, 0}),
  triangleColor={175, 175, 175})
```

the visual representation of Figure 20 of the Prismatic joint is achieved.

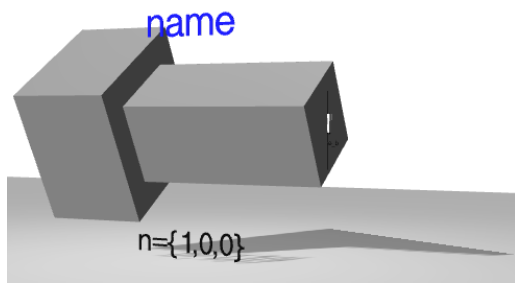


Figure 20. Prismatic joint using triangular mesh

The calls to the function `Box` return 12 triangles each. These are translated and combined by the CSG union operator to a new triangular mesh representing the union. Finally this solid is rotated. A default argument might be introduced, in addition to the mesh argument, with a 2D standard representation for those tools that does not support 3D primitives.

In addition to this triangular mesh primitive extending the visual representation to 3D, also the coordinate system need to be extended, for example:

```
coordinateSystem(
  extent3D={{-100,-100,-100},{100,100,100}})
```

Furthermore, the Placement annotations of components need to be extended to 3D positioning, sizing and orientation, for example (with `qx`, `qy` and `qz` representing the direction of the local `x`, `y` and `z` axis):

```
Placement(transformation3D(
  position={-50,50,10},
  size={10,10,10},
  rotation={qx, qy, qz}))
```

The position and rotation of connectors in 3D is especially important to enable assembly operations where the connected components immediately rotate to be correctly assembled.

7 Playmola Architecture

Playmola is designed to run in a web browser and it communicates with Dymola using remote JavaScript calls encoded in JSON format. Dymola 2016 version (Dassault Systèmes, 2015) has support for remote Modelica function calls performed in Java, JavaScript or Python. This API has been extended by certain new functions.

Three.js (Dirksen, 2013), (three.js, 2015), a popular JavaScript 3D rendering API based on WebGL, was chosen for Playmola's development since it provides the required graphical functionality. Dymola was enhanced to include functions for exporting 3D models in a format compatible with three.js. The built-in VRML loader was extended with extra functionality to be able to load 3D models in VRML format.

Playmola imports data from Dymola by the means of a client/server relationship. This architecture is used in order to load component models defined in Dymola into Playmola, to send the parameterized component instances back to Dymola in the form of generated Modelica code, for performing simulation and to query frame data for the result animation.

jQuery Mobile (jQuery Foundation, 2015) was used for GUI elements and input handling, which, together with the cross-platform nature of HTML5, allows Playmola to be used on multiple devices. The prototype was developed on browsers running on a PC, but also runs on iPad (Figure 21) and other touch-based devices.



Figure 21. Playmola on the iPad

8 Stereo Viewing and Virtual Reality

In order to become more interesting and allow a 3D experience, Playmola can be run in stereo mode for 3D viewing. The rendering package used, Three.js, supports side-by-side rendering of the scene.

Figure 22 shows running Playmola on an iPhone, transferring this stereo image to a 3D TV using Apple TV. The TV is set to stereo side-by-side mode and active 3D glasses are used for viewing the 3D content.

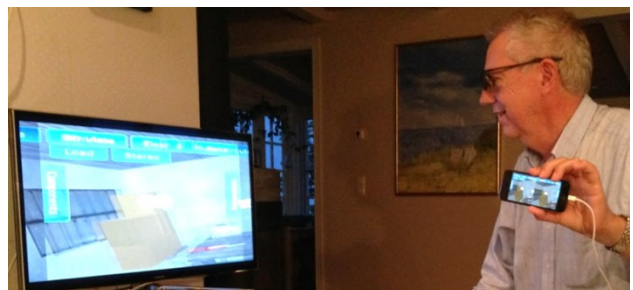


Figure 22. Stereo image on iPhone and 3D TV

The stereo side-by-side rendering is also what is needed for virtual reality gear such as Oculus Rift or the low cost Google Cardboard. Google Cardboard

mounts a smart phone in front of your eyes. Head motion is tracked and controls the viewing. It is possible to make compositions by looking at an object, pressing the button on the Cardboard (Google Cardboard V2 is needed for iPhone), looking in the direction of the destination and releasing the button, Figure 23. The white circle in the middle of the view represents the focus point.



Figure 23. Google Cardboard and side-by-side projection

We have tested the Leap Motion controller that tracks your fingers. However, our initial assessment is that the resolution is not sufficient for gripping objects when authoring models.

We believe Cardboard will revolutionize education. It is then important that the teacher and the students can be in the same virtual reality. We are currently investigating technologies to enable sharing the same scene. It means that all clients communicate with a scene server that broadcasts any changes to the scene.

As the next step, we are eager to start testing HoloLens from Microsoft since it supports *augmented reality*. It means that the teacher and the students can be seen in the virtual model and can act on it.

9 Future Work

The work is continuing and this section will summarize some of the features in the pipeline.

Traditionally, a mechanism assembly is done by putting joints between frames of bodies. However, it makes sense from an object oriented point of view to associate more information with the bodies. For example, a bar that has a cylindrical hole is prepared for mating with another body with a hole, i.e. it is prepared to have one rotational and optionally one translational degree of freedom. By associating this information with the body, the tool can automatically insert either a Revolute or a Cylindrical joint when connected to another body. Standardized base classes with predefined properties for such *smart connectors* will be used to store such information about degrees of freedom.

In order to better support direct manipulation and virtual reality authoring without keyboard, we are

working on widgets for positioning, rotation, scaling and setting DOFs. Figure 24 shows the positioning tool which is inspired by such a tool in Unity.

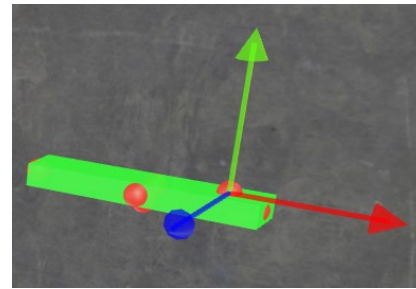


Figure 24. Positioning tool

Orientation in 3D is complex to comprehend for students. However, it is important when building and initializing complex mechanisms such as a Stewart platform. Therefore support should be given in Playmola.

The Turtle graphics methodology is a good way of learning 2D computer graphics. It is based on two commands, *forward* a certain distance and *rotate* a certain angle plus *repetition*. For *3D Turtle graphics*, two rotation commands are needed such as Roll and Turn (Verhoeff, 2009). Roll is rotation around the x-axis and Turn around the z-axis.

By using 16 bars and connectors with Roll($\pm 90^\circ$) and Turn(130.06°), it is possible to build a regular 3D polygon as shown in Figure 25 (Verhoeff, 2009).

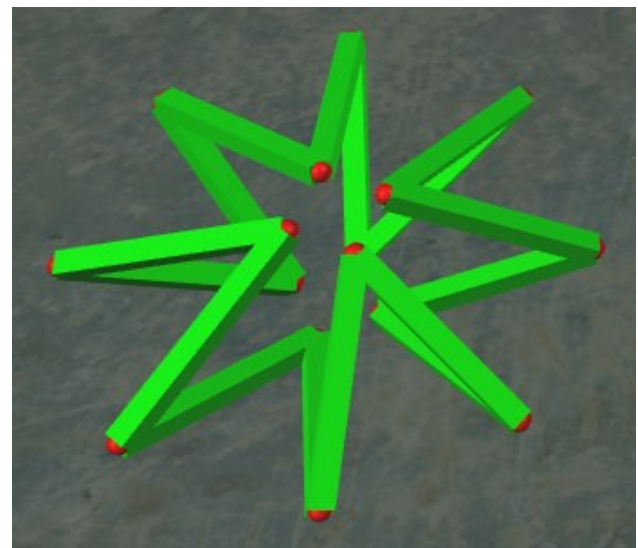


Figure 25. Regular 3D polygon

In order to build such regular structures we plan a Repeat Paste command which pastes a certain number of levels and makes the connections.

Playmola will also allow the inclusion of *plots* in the environment, i.e. on the walls, on the computer or on

the desk. The plots are regular 3D object although flat, i.e. they have position, orientation and scale 3D transformation as shown in Figure 26.

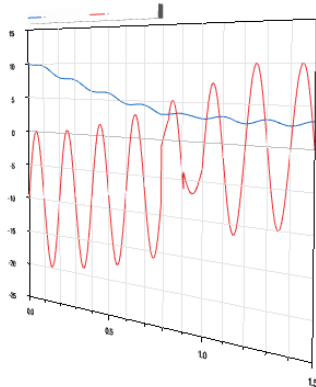


Figure 26. Plot as 3D object

3D is not only important for mechanical systems. We have started to experiment on how to handle *fluid systems*. Interestingly, some of the data associated with a mechanical connector, i.e. position and orientation are also needed for fluid components such as tanks and reactors. For pipes, also the routing is important.

Regarding animation of fluid flows and their properties and electrical flows, we will use the particle simulations available in packages such as three.js. The speed of particles corresponds to the flow rate and color can be used for temperature as shown in Figure 27. The level in a vessel can conveniently be handled by the shaders of the GPU by making the vessel transparent and filling the vessel up to a certain level when rendering the fluid.

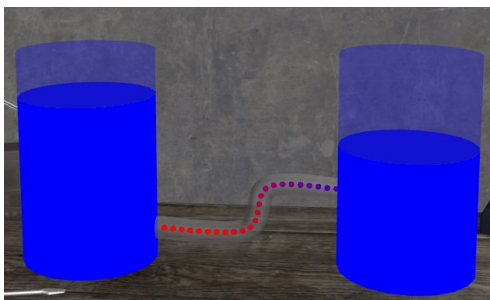


Figure 27. Flow and level animation.

10 Conclusions

A completely different and more modern 3D representation of Modelica models is proposed. It will help professionals to comprehend complex models and is more attractive, intuitive and fun for new-comers.

These ideas have been prototyped in a web application which uses Dymola in server mode for creating 3D representations and for performing simulations.

The support for low cost virtual reality gear such as Google Cardboard will enable faster adaption of this kind of modeling and simulation techniques in education.

Acknowledgements

This work has partly been performed as a bachelor degree project at Malmö University. The first author served as an industrial advisor and Olle Lindeberg as the formal supervisor.

The authors want to thank Carl Fredrik Abelson for extending Dymola with needed remote JavaScript calls for various functionalities and for exporting WebGL code for 3D Modelica representations.

The authors also want to thank Martin Malmheden, who has been heavily involved in the discussions on how to continue the work after the bachelor degree project, for his contributions.

References

- Algodoo (2015): <http://www.algodoo.com/>
- Bijl, Jonatan L. and Boer, Csaba A. (2011): Advanced 3D Visualization for Simulation Using Game Technology. In: Proceedings of the Winter Simulation Conference. WSC'11. Phoenix, Arizona: Winter Simulation Conference, 2011, pp. 2815–2826.
- Dassault Systèmes (2015): Dymola 2016. <http://www.Dymola.com>
- Deterding, Sebastian. (2011) Situated motivational affordances of game elements: A conceptual model. In: CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.
- Deterding, Sebastian et al. (2011): From Game Design Elements to Gamefulness: Defining “Gamification”. In: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments. MindTrek '11. New York, NY, USA: ACM, 2011, pp. 9–15
- Dirksen, Jos (2013): Learning Three.js: The JavaScript 3D Library for WebGL. Packt Publishing (October 2013)
- Elmqvist H., Goteman A., Roxling V., Ghandriz T. (2015): Generic Modelica Framework for MultiBody Contacts and Discrete Element Method. Proceedings 11th International Modelica Conference, Versailles, September 21-23, 2015.
- Hamari, J., Koivisto, J., and Sarsa, H. (2014): Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. In: System Sciences (HICSS), 2014 47th Hawaii International Conference on. Jan. 2014, pp. 3025–3034.
- jQuery Foundation (2015): jQuery Mobile <https://jquerymobile.com> (visited on 05/16/2015).
- Malone, Thomas W. (1982): Heuristics for Designing Enjoyable User Interfaces: Lessons from Computer Games. In: Proceedings of the 1982 Conference on Human Factors in Computing Systems. CHI '82. New York, NY, USA: ACM, 1982, pp. 63–68.
- Nicholson, Scott. (2012): A User-Centered Theoretical Framework for Meaningful Gamification. In: Proceedings

of Games+Learning+Society 8.0. Madison, WI, USA, 2012.

Podolefsky, Noah S., Perkins, Katherine K., and Adams, Wendy K. (2010): Factors promoting engaged exploration with computer simulations. In: Phys. Rev. ST Phys. Educ. Res. 6.2 (Oct. 2010), p. 020117

Robertson, Margaret. (2010): Can't play, won't play. <http://hideandseek.net/2010/10/06/cant-play-wont-play/> (visited on 04/07/2015).

Sabourin, J.L. and Lester, J.C. (2014): Affect and Engagement in Game-Based Learning Environments. In: Affective Computing, IEEE Transactions on 5.1 (Jan. 2014), pp. 45– 56.

Seaborn, Katie and Fels, Deborah I. (2015): Gamification in theory and action: A survey. In: International Journal of Human-Computer Studies 74 (2015), pp. 14–31.

three.js (2015): <http://threejs.org/> (visited on 04/12/2015).

Verhoeff T., Verhoeff K. (2009): Regular 3D Polygonal Circuits of Constant Torsion. Bridges 2009: Mathematics, Music, Art, Architecture, Culture. <http://archive.bridgesmathart.org/2009/bridges2009-223.pdf>

Wieman, Carl E. and Perkins, Katherine K. (2006): A powerful tool for teaching science. In: *Nat Phys* 2.5 (May 2006), pp. 290-292.

Holistic Virtual Testing and Analysis of a Concept Hybrid Electric Vehicle Model

Jonathan Spike¹ Dr. Johannes Friebe¹ Dr. Chad Schmitke¹
Dr. Christian Donn² Michael Folie² Valerie Bensch²
Christine Schwarz³

¹Maplesoft, Waterloo, Ontario, Canada, {jspike, jfriebe, cschmitke}@maplesoft.com

²IPG Automotive GmbH, Karlsruhe, Germany,
{christian.donn, michael.folie, valerie.bensch}@ipg.de

³ISKO engineers AG, Leonberg, Germany, Christine.Schwarz@isko-engineers.de

Abstract

In this paper; the development, integration, and analysis of a hybrid electric vehicle (HEV) using system level virtual test will be presented. The work will discuss how a Modelica-based Parallel HEV powertrain model developed using MapleSim™ is integrated into industrial vehicle modeling software tool (IPG CarMaker®) using the Functional Mockup Interface (FMI) standard; and how, using API commands, virtual testing and analysis was performed with an optimization tool (Noesis Optimus®). The acausal modeling of the HEV powertrain was done using Modelica 3.2.1, allowing the flow of energy to be inferred from the operating characteristics and controller design. The multidomain model uses components from the electrical and mechanical libraries, including commercialized library components from MapleSim's Driveline Component Library and Battery Component Library.

Keywords: Hybrid Electric Vehicle, Powertrain, FMI, Driveline, Modelica, MapleSim, CarMaker, Optimus

1 Introduction

Powertrain hybridization assists an internal combustion engine to operate with optimal efficiency and enables the recuperation of kinetic energy during braking. This increases a vehicle's fuel efficiency and reduces its exhaust emissions. Additionally, powertrain electrification offers many possibilities for increasing longitudinal and lateral vehicle dynamics (Appel, Sterzing-Oppel, *et al.*, 2015). However, in view of the wide range of variants and concepts of hybrid electric vehicles, finding optimized setups often poses a challenge due to the varying boundary conditions, different cases of application, as well as interdependent vehicle subsystems.

Although the process starts with simulation runs to investigate vehicle concepts and operating strategies using different powertrain topologies or components, it is crucial to examine the performance of the overall system, as well as the functionality and interaction of all relevant subsystems, in realistic scenarios and

conditions in order to meet the final development targets. This is where optimization processes and tools can assist – to find the best compromise, taking into account all the various design constraints.

In this paper, an open integration and test platform is used for the multi-objective optimization of the powertrain concept of a hybrid vehicle. This was done for different driving scenarios and driver types while taking into account longitudinal and lateral vehicle dynamics. A comparative study of fuel efficiency and performance for a hybrid-electric powertrain with different battery sizes and operating strategies was carried out, using the FMI approach to integrate a detailed vehicle powertrain model into a comprehensive full-vehicle model driven by a virtual driver on a virtual road.

2 Modeling and Simulation Environment

The comparative study of fuel efficiency and performance of different hybrid electric vehicle powertrain concepts carried out in this work was conducted using three development tools. The powertrain model was developed in MapleSim™, the multidomain modeling and simulation environment from Maplesoft™. It was then converted into a Functional Mockup Unit (FMU) for integration with the test environment using IPG CarMaker®, an open integration and testing platform (Kobayashi, Donn, 2015). The virtual vehicle, road and driver were set up using CarMaker, which then linked the entire virtual environment with Optimus®, a Process Integration and Design Optimization (PIDO) platform from Noesis, to perform comprehensive multi-objective optimization. Figure 1 shows an overview of the modeling and simulation environment.

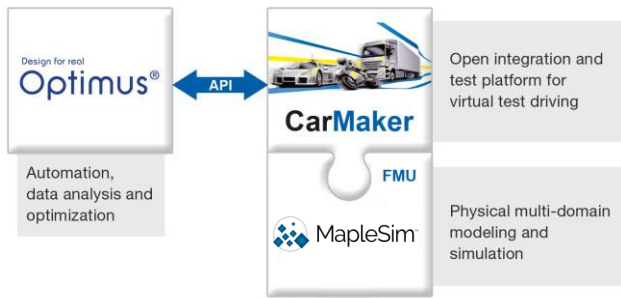


Figure 1. Modeling and Simulation Environment

3 Powertrain Model

The HEV powertrain model was developed using MapleSim’s Modelica-based library components. The Modelica physical modeling language has become widely-used and allows multidomain modeling within one model structure (Otter, Elmquist *et al*, 2007). The HEV powertrain model utilized MapleSim’s own Driveline Component Library, Battery Component Library, as well as custom components to generate the model. The intuitive modeling process allows dragging-and-dropping of predefined components, connecting them together, and then specifying the operating parameters to obtain preliminary results. The model was then converted using the FMU code generation template for integration with vehicle modeling tools that comply with the FMI standard, such as IPG CarMaker. The automatically-generated code includes significant optimizations obtained by applying symbolic techniques. This results in fast execution time, making it suitable for both real-time applications and optimization solutions. The FMI export tool, used to integrate the HEV powertrain with CarMaker, provides the ability to generate FMI 1.0 or 2.0 along with both Model Exchange (FMU that uses the solver in the target environment) and Co-Simulation (FMU with embedded solver) configurations. The HEV powertrain model details are specified in the following section.

3.1 Reference Model

In order to meet the goal of a realistic HEV powertrain with a continuously variable transmission (CVT) and maximizing efficiency, a commercially-available powertrain concept was selected. The selected powertrain configuration is a Parallel HEV model with a CVT, inspired by the 2006 Honda Civic Integrated Motor Assist (IMA) powertrain (Hofman, Druten *et al*, 2005). The Honda IMA configuration uses the electric motor (EM) and internal combustion engine (ICE) in a parallel configuration therefore driving the same driveshaft. By adjusting the operating behavior of both the EM and ICE, an energy balance may be determined

to maximize efficiency. The EM connected in this configuration can provide engine balancing, function as the starter motor, and allow for energy flow to and from the driveshaft for either motor assisting or electrical regeneration. This parallel HEV powertrain configuration allows for six different modes of operation (Donn, Folie *et al*, 2015):

1. **Engine Driving (E):** The ICE is the only source for providing the power
2. **Motor Assist (MA):** The ICE and EM both deliver the requested driving power
3. **Motor Driving (M):** The EM is the only source for providing the power
4. **Charging (CH):** The ICE deliver power for both the vehicle and battery charge power demand.
5. **Brake Energy Recovery (BER):** During deceleration, part of decelerating power is recovered by the EM and stored in the battery
6. **Idle Stop (IS):** The ICE is stopped during full vehicle stops. Meanwhile, the auxiliaries are powered from the battery.

3.2 Powertrain Component Modeling

The EM and ICE driveshaft is also coupled to the input of the CVT. The output from the CVT then drives the wheels as the torque is transferred through a differential. Since the EM motor is capable of both providing energy and extracting energy from the driveshaft, a Power Electric Controller (PEC) system is required to manage the energy flow to and from the EM and the battery. Figure 2 presents the energy flow between the different powertrain subsystems.

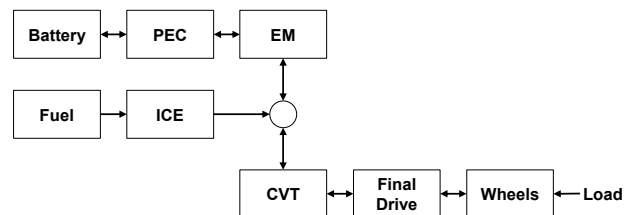


Figure 2. Energy Flow Diagram for a Parallel Hybrid Electric Vehicle Powertrain using a CVT.

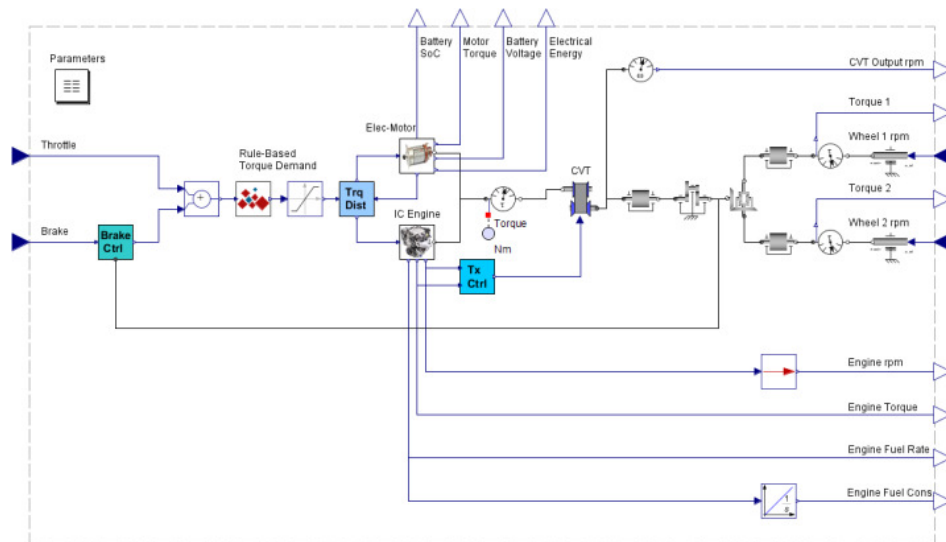


Figure 3. FMU Generation Configuration of Hybrid Electric Vehicle Powertrain with a CVT

The acausal physical component modeling characteristics of the Modelica components allow the powertrain system to be developed based on the energy flow diagram of Figure 2. The fundamental physical equations within the Modelica components allow the system to provide/extract energy at the wheels as the vehicle is accelerating or decelerating without requiring special modeling structures or definitions for each component use case. Figure 3 shows an overview of the HEV powertrain model prepared for FMI export used in this work. The FMU uses input/output signals, unlike the physical ports in the Modelica components, thus requiring the model to be prepared in the form that is compatible with the FMI standard.

The MapleSim model was generated based on limited technical data for the 2006 Honda Civic IMA powertrain. Model data was obtained from multiple sources, and estimates were used when information was not available. Several of the key components (ICE, EM, Battery, PEC) were not explicitly available and approximate models were used to replicate the desired behavior. The following specifies additional details for each of the major subsystems.

ICE System: The ICE subsystem utilized engine performance data for a 1.36L engine. The engine map was implemented using MapleSim's Driveline engine component that related the engine throttle to the speed-torque tabular data. The ICE model was configured to allow the engine to be used with a stop-start system. The ICE model also provides instantaneous fuel rate, for fuel consumption calculation.

EM, PEC, and Battery Systems: The electric motor subsystem housed all three components (EM, PEC, and the battery). The EM selected was a permanent magnet DC motor with a rated power of 13.8kW. By using the DC-EM the PEC unit and control strategy can be relatively simple, as the PEC was not a primary focus for this investigation. However, the PEC did include features to protect the battery from being over charged or fully discharged and appropriately limit the electrical current. The battery model selected was the nickel-metal hydride (NiMH) equivalent circuit battery model from the MapleSim Battery Component Library. Further discussion on the battery is contained in Section 3.3.

Transmission System: The CVT component was directly obtained from MapleSim's Driveline Component Library. This component allows the transmission ratio to be adjusted based on the specified gear ratio. The CVT gear ratio operates within the continuous range of 0.45 and 2.6. The CVT was coupled to a final drive system allowing for further gear reduction. The control approach used with the CVT and ICE is discussed in Section 3.4.

3.3 Battery Pack

The Nickel Metal Hydride (NiMH) battery used in the reference vehicle powertrain provides the electrical energy storage required for the HEV and an additional electrical charge to operate the auxiliary components that would normally operate the 12V electrical systems. This battery supplies power during electrical motoring (M) or motor assist (MA) modes. During the charging (CH) or braking (BER) operating modes the energy is stored in the high voltage battery pack.

The NiMH battery model was selected from the MapleSim Battery Component Library to resemble the battery used in the reference vehicle. The battery library includes additional battery chemistry such as Lead Acid and Li-ion, each with available electro-chemical and equivalent circuit models. The NiMH battery model used was an equivalent circuit model that implements nonlinear resistors and capacitors. The values of the resistors and capacitors are determined as a nonlinear function of battery state-of-charge (SoC). Figure 4 illustrates the underlining equivalent circuit structure representing the battery. Equation (1) defines the exponential-polynomial function used (Chen, Rincón-Mora, 2006).

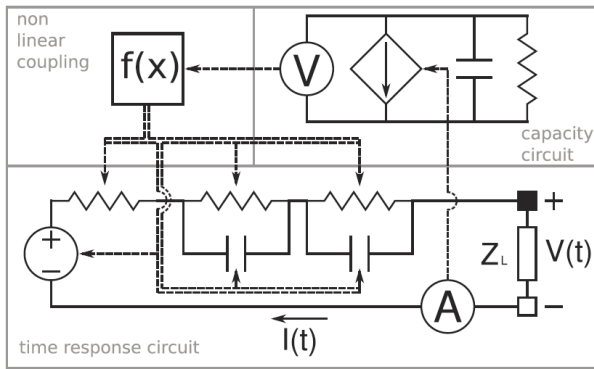


Figure 4. Equivalent Circuit Model Diagram for the NiMH Battery

$$\begin{aligned}
 R_i &= k_{1,Ri} \exp(k_{2,Ri}SoC) + k_{3,Ri} + k_{4,Ri}SoC \\
 &\quad + k_{5,Ri}SoC^2 + \dots \\
 C_j &= \frac{1}{R_j} (k_{1,Ri} \exp(k_{2,Ri}SoC) + k_{3,Ri} + k_{4,Ri}SoC \\
 &\quad + k_{5,Ri}SoC^2 + \dots) \\
 i &= \{0,1,2\}, \quad j = \{1,2\}
 \end{aligned} \quad (1)$$

In order to prevent the battery from reaching non-physical conditions, the battery model can terminate the simulation if the battery is discharged past a minimum level or similarly if the battery is over charged. As a result the PEC implemented is tasked with maintaining the battery within the desired SoC range ensuring the proper battery operation.

The calculating the instantaneous battery SoC is determined using Equation (2).

$$SoC(t) = \frac{Q(t)}{Q_{max}} \quad (2)$$

SoC(t) and Q(t) are the instantaneous state of charge and battery electrical change, while Q_{max} is the maximum electrical charge that the battery pack is capable of maintaining. Therefore, the value of SoC ranges between 1 (fully charged) and 0 (fully

discharged). More details about the battery model can be found in (Dao, Vyasarayani *et al*, 2012)

3.4 Powertrain Control Strategy

The hybrid powertrain energy management strategy determines the distribution of power between the EM, ICE and the battery while fulfilling the instantaneous power demand requirements. The implemented control strategy has a significant impact on the useable energy and performance of the powertrain (Donn, Folie *et al*, 2015). Figure 5 illustrates the energy flow chart, defining the block subsystems used in the control strategy. The powertrain is combined with the vehicle and driver model in which case the applied torque can either accelerate the vehicle or apply regenerative braking depending on the driver pedal input command.

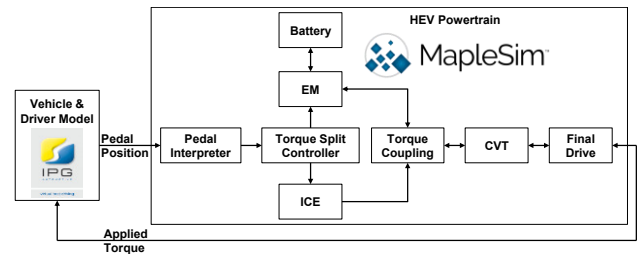


Figure 5. Energy Management Flow Chart for the Parallel Hybrid Electric Vehicle

The powertrain's torque demand (T_d) is calculated by converting information from the positions of the accelerator and brake pedals. The combined torque output from the EM and ICE is used to meet the torque demand. As a result, the distribution between the two systems – as determined by the torque split controller – provides an important part of the control strategy. In this work the torque split controller is a rule-based approach, determined by the torque demand and battery SoC.

Figure 6 illustrates an extract of the sample code that was used to define the custom Modelica torque splitter component. As a result, the powertrain operating mode is determined by the torque splitter output. When the battery SoC is at a sufficient operating point, the powertrain operating characteristics are as defined in Table 1. Otherwise, when the battery SoC is below the 10% minimum specified threshold, the torque splitter will ensure not to load the EM. Instead, the ICE will be expected to provide all of the accelerating torque demand.

```

1 model TorqueSplit
2   Modelica.Blocks.Interfaces.RealInput Td "Torque Demand"
3   annotation(*...*);
4   Modelica.Blocks.Interfaces.RealOutput TdENG "Torque Demand for Electric Motor"
5   annotation(*...*);
6   Modelica.Blocks.Interfaces.RealOutput TdEM "Torque Demand for Electric Motor"
7   annotation(*...*);
8   Modelica.Blocks.Interfaces.RealInput SOC "Battery State of Charge"
9   annotation(*...*);
10
11   parameter Real SOCmin = 0.1 "Minimum Battery State of Charge";
12   parameter Real RegenTrq = 15 "Electric Motor Regenera-tive Torque Load";
13   parameter Real TE = 25 "Torque Demand: Battery Regeneration Lower Limit ";
14   parameter Real TMA = 100 "Torque Demand: Motor Assist Starting Point";
15   parameter Real TRg = 65 "Torque Demand: Battery Regeneration Upper Limit ";
16   parameter Real ElecMARRatio = 0.5 "Motor Assist Torque Distribution Ratio";
17
18 equation
19   TdEM =
20     if SOC < SOCmin and 0 < Td then 0
21     else if Td < 0 then Td
22     else if 0 <= Td and Td < TE then Td
23     else if TE <= Td and Td < TRg then -RegenTrq
24     else if TRg <= Td and Td < TMA then 0
25     else if TMA <= Td then ElecMARRatio * (Td - TMA)
26     else Td;
27   TdENG =
28     if SOC < SOCmin and 0 < Td then Td
29     else if Td < 0 then 0
30     else if TE <= Td and Td < TRg then Td - TdEM
31     else if TRg <= Td and Td < TMA then Td
32     else if TMA <= Td then Td - TdEM
33     else 0;
34 end TorqueSplit;

```

Figure 6. Modelica Code Sample of the Torque Splitter

Table 1. Powertrain torque split strategy

Mode	Torque Demand Range	Torque Split
BER	$T_d < 0$	$T_{ICE} = 0$ $T_{EM} = T_d - T_{Drag}$
M	$0 < T_d < TE$	$T_{ICE} = 0$ $T_{EM} = T_d$
CH	$TE < T_d < TRg$	$T_{ICE} = T_d + T_{EM}$ $T_{EM} = RegTrq$
E	$TRg < T_d < TMA$	$T_{ICE} = T_d$ $T_{EM} = 0$
MA	$T_d > TMA$	$T_{ICE} = T_d - T_{EM}$ $T_{EM} = 0.5(T_d - TMA)$

Once the torque distribution has been determined, the engine and transmission control systems are required to operate together. The approach selected assumes that the operating performance of the ICE is the major focus. The ICE and transmission control approach uses the minimal Brake Specific Fuel Consumption (BSFC) (Bai, Maguire *et al*, 2013). Both the engine map and optimal BSFC operating points have been determined in a previous analysis.

The powertrain utilizes two controllers that continuously regulate the engine throttle and the transmission ratio, adjusting the ICE instantaneous operating point to the BSFC optimal operating point. Additionally, when the vehicle is running at speeds below 20Km/h, the electrical regenerative braking is not used. Rather, only the hydraulic brake (in the CarMaker brake model) is used to stop the vehicle.

4 Process Integration and Optimization

The Optimus software tool provided an automated approach to performing a large number of simulations with parameter configuration changes. Optimus takes over the task of manually running simulations, obtaining, storing, and then analyzing the results. This automated approach is performed through direct communicating with CarMaker using API functions (alternatively the open ASCII interface communication can be used). Figure 7 shows the workflow deployed with the Optimus tool. Starting on the left, different parameter sets are selected, and substituted into CarMaker for simulation. Following that, the results are processed. For example, the total energy used in the simulation is determined based on the total electrical energy and fuel consumed during the entire simulation run.

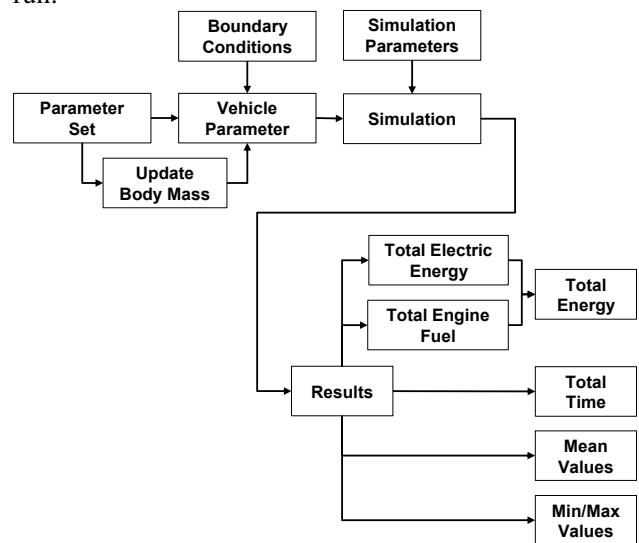


Figure 7. Optimus Workflow of the Automated Process

One vehicle parameter, the vehicle body mass, was determined as a result of adjusting the capacity of the HEV powertrain battery. The base vehicle body mass of 1301kg is updated with the additional battery mass calculated based on (Noshin, Verbrugge *et al*, 2010) using an energy density of 80Wh/kg. Once the simulation is complete, the results can be used as inputs for an objective function, or as constraints during an optimization process.

4.1 Design of Experiments

The design process, particularly for systems with many undetermined parameters, has the challenge of defining a suitable parameter set to use for optimization. This results in a large design space during the optimization process. One automated approach to reducing the design space is to perform a sensitivity analysis. The sensitivity analysis provides a method of determining the design variables that have the most significant

influence on the response. The most significant variables can be used in further analysis while simultaneously reducing the design space by excluding less significant factors.

A sensitivity analysis study was performed to examine which of the parameters had the greatest influence on the response of the entire system performance. Figure 8 illustrates the results of the Latin Hypercube sampling with 100 experiments per traffic-light and driver-behavior configuration. One finding noted in the sensitivity analysis was that the traffic light timing considerably influenced simulation results, most notable for the defensive driver. This can be observed clearly by the vertical lines in the plot results with the regular traffic light control. To ensure each simulation would experience the same influence from each traffic light, the traffic lights were adjusted to ensure the vehicle would stop as it approached the intersection. The adjusted traffic light control plot points are obtained using this new traffic light timing. It can be noted that the aggressive driver was able to reliably complete the simulation run in less time than the defensive driver with the tradeoff appearing in higher total energy consumption. The adjusted traffic light control will be used for further discussion in this work.

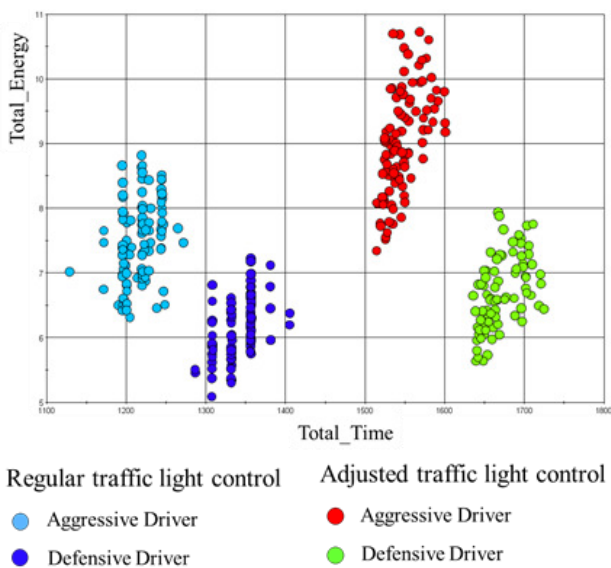


Figure 8. DOE Results from Drive Cycle with Different Traffic Light Control and Driver Behavior

A set of six parameters were selected to be used in the optimization process with an interval range that remained within physically feasible limits. Table 2 shows the six parameters and the ranges selected to drive the optimization process. It also includes the vehicle body mass parameter determined by the battery capacity selected.

Table 2. Optimization Input Parameters

Parameter	Description	Min Value	Max Value
BatCap	Capacity of Battery	10 [Ah]/ 1.44 [kWh]	150 [Ah]/ 21.6 [kWh]
TE	Torque Demand: Battery Regeneration Lower Limit	10 [Nm]	50 [Nm]
TRg	Torque Demand: Battery Regeneration Upper Limit	25 [Nm]	100 [Nm]
TMA	Torque Demand: Motor Assist Starting Point	75 [Nm]	150 [Nm]
RegenTrq	Electric Motor Regenerative Torque Load	5 [Nm]	50 [Nm]
FinDrRatio	Final Drive Ratio	4 [-]	7 [-]
Body_mass	Vehicle Mass	1319 [kg]	1571 [kg]

The Latin Hypercube 100 sample experiments were used to determine the influence of selected parameters on the total response time and total energy consumption. Table 3 shows the output parameters monitored during the experiments.

Table 3. Optimization Output Parameters

Parameter	Description	Unit
Total_Time	Total Time for Drive Cycle	[s]
Total_Energy	Total Energy Consumption	[kWh]

The correlation denoted in Table 4 represents both the Pearson and Spearman rank correlation coefficients. For both, values close to +/- 1 indicate a significant linear or monotonic correlation between the input parameter and the output parameter. For example the input parameter TMA has been determined to have strong correlation with the total energy consumption particularly for the defensive driver.

Table 4. Correlation table for the adapted traffic light control drive cycle

Driver Type	Parameter	Total Time Pearson (Spearman)	Total Energy Pearson (Spearman)
AGGRESSIVE DRIVER	BatCap	0.015 (0.205)	0.433 (0.434)
	TE	0.173 (0.191)	-0.333 (-0.308)
	TRg	-0.137 (-0.127)	0.196 (0.188)

	TMA	0.331 (0.682)	0.682 (0.683)
	RegenTrq	0.097 (0.048)	0.327 (0.323)
	FinDrRatio	-0.053 (-0.259)	0.083 (0.072)
DEFENSIVE DRIVER	BatCap	0.098 (0.193)	0.576 (0.574)
	TE	0.082 (0.118)	-0.345 (-0.331)
	TRg	-0.083 (-0.041)	0.245 (0.223)
	TMA	0.318 (0.723)	0.521 (0.524)
	RegenTrq	0.107 (0.076)	0.320 (0.322)
	FinDrRatio	-0.068 (-0.227)	0.121 (0.107)

4.2 Multi-objective Optimization

The selection of a suitable strategy and algorithm for optimization can be influenced by multiple factors. Aside from parallelization possibilities and computer resources, the number of design variables and system behavior can have a great influence on the strategy deployed. In this case, the simulation time was not a limiting factor - with the simulation performance approximately 8-fold faster than real time. Additionally, because the HEV system under consideration is highly nonlinear, a global optimization algorithm such as an evolutionary strategy should be chosen.

This work considered two simultaneous objective functions: minimize drive cycle total time and minimize energy consumption. These two conflicting objectives will result in more than one optimal solution. As a result a Pareto front is determined defining the complete set of compromised optimal solutions. The evolutionary algorithm NSEA+ (Non-dominant Sorting Evolutionary Algorithm) was used to detect the Pareto points for the Nordschleife drive cycle. Figure 9 and Figure 10 show the optimal results for both the aggressive and defensive driver. The optimization process also considered the constraints $TE < TRg$ and $TRg < TMA$ as part of the requirement in the HEV powertrain controller.

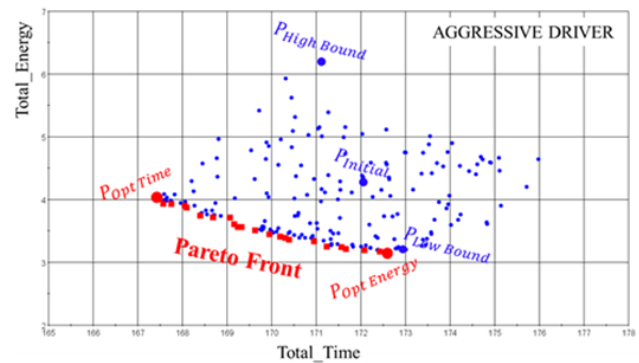


Figure 9. Optimization Results: Nordschleife Drive Cycle and Aggressive Driver Behavior

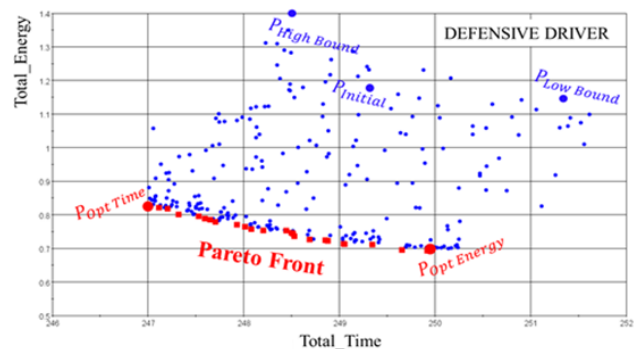


Figure 10. Optimization Results: Nordschleife Drive Cycle and Defensive Driver Behavior

The five unique points denoted in Figure 9 and Figure 10 are defined as follows:

1. P_initial: the initial design response. P_initial was determined by using the mean value for each design variable considered in the optimization process.
2. P_Opt_Time: the optimal solution considering the optimal time objective only
3. P_Opt_Energy: the optimal solution considering the optimal energy consumption objective only
4. P_High: response with all design space variable using upper boundary values
5. P_Low: response with all design space variable using lower boundary values

For each of the objectives functions a considerable improvement was made compared to the initial design variable selection. It is worth noting that some of the optimal results were found at the boundaries, suggesting that increasing the boundary limits might further improve the optimal results.

Table 5. Optimization Results for Nürburgring-Nordschleife Drive Cycle

	BatCap [Ah]	TE [Nm]	TR _g [Nm]	TMA [Nm]	RegenTrq [Nm]
P-Low	10	10	25	75	5
P-Initial	80	30	62.5	112.5	27.5
P-High	150	50	100	150	50
Aggressive Drive P _{Opt Time}	11.9	31.5	43.4	76.0	25.0
Aggressive Drive P _{Opt Energy}	10.2	43.6	49.8	75.9	10.1
Defensive Driver P _{Opt Time}	16.7	49.9	52.1	75.5	20.6
Defensive Driver P _{Opt Energy}	17.4	50.0	51.1	75.7	22.1
	FinDRatio [-]	Body_Mass [kg]	Total Energy Consum. [kWh]	Total Time [s]	
P-Low	4	1319	1.15	251.4	
P-Initial	5.5	1445	1.17	249.3	
P-High	7	1571	1.46	248.6	
Aggressive Drive P _{Opt Time}	6.8	1322	3.93	167.8	
Aggressive Drive P _{Opt Energy}	4.3	1319	3.17	172.4	
Defensive Driver P _{Opt Time}	6.1	1331	0.79	247.6	
Defensive Driver P _{Opt Energy}	4.9	1332	0.72	248.9	

This example showed that a hybrid powertrain with a smaller battery was best for the aggressive driver on this particular driver cycle. As for the defensive driver, a higher torque limit (TE) for pure electric mode was more beneficial.

5 Conclusion

In this paper, a new and efficient approach for optimizing the design parameters of a complete powertrain model with a hybrid electric vehicle was shown. The goal was not only to integrate a complex powertrain model in an easy and intuitive way for a given vehicle, but also to detect coherences and perform advanced optimizations using virtual test driving in an automated loop without much user effort.

The powertrain model was created using MapleSim. It is a Parallel HEV model with a CVT, inspired by the 2006 Honda Civic IMA powertrain, and allows for six modes of operation. It was integrated with the CarMaker vehicle model using the Functional Mockup Interface (FMI) standard, and simulated under two scenarios – representing defensive and aggressive driver behavior. Design of experiment and multi-variable optimization were performed using Optimus.

While this work was focused on investigating the method and workflow of combining the different tools, the realistic results that were achieved indicate that this is an effective method for investigating and optimizing a hybrid powertrain concept. The combined effect of the different tools creates an easy-to-use and powerful environment for comprehensive development and optimization of a hybrid electric powertrain concept. The automated evaluation of multiple simulations allows good insight into a highly complex system to understand its dependencies. Applying evolutionary optimization algorithms helped to find the optimal settings required to meet pre-defined performance goals.

References

- C. Appel, S. Sterzing-Oppel, J. Gerstenberg, C. Donn. Comprehensive and crossdomain vehicle simulation for the electrification of sports cars. Graz Symposium Virtual Vehicle, Graz, 2015.
- S. Bai, J. Maguire and H. Peng. Dynamic Analysis and Control System Design of Automatic Transmissions, Warrendale, Pennsylvania, USA: SAE International, 2013.
- M. Chen and G.A. Rincón-Mora. Accurate electrical battery model capable of predicting runtime and I-V performance. *IEEE Transactions of Energy Conversion*, Vol. 21, No. 2, 2006.
- T.-S. Dao, C.P. Vyasrayani, J. McPhee. Simplification and order reduction of lithium-ion battery model based on porous-electrode theory. *Journal of Power Sources*, 01/2012, page 329–337, DOI: 10.1016/j.jpowsour.2011.09.034, 2012.

- C. Donn, M. Folie, V. Bensch, J. Friebe, J. Spike, P. Goossens and C. Schwarz. Concept analysis & system design of a hybrid electric vehicle with virtual test driving. 15th Stuttgart International Symposium Automotive and Engine Technology, Stuttgart, 2015.
- T. Hofman, R.M. van Druten, A.F.A Serrarens and J. van Baalen. A fundamental case study on the Prius and IMA drivetrain concepts. 21st Worldwide International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium (EVS-21), Monaco, 2005.
- M. Kobayashi and C. Donn. Fusion of simulation and testing within the automotive development process using an open integration and test platform. 2015 JSAE Annual Congress, Yokohama, 2015.
- O. Noshin, B. Verbrugge, G. Mulder, P. van den Bossche, J. van Mierlo, M. Daowd, M. Dhaens and S. Pauwels. Evaluation of performance characteristics of various lithium-ion batteries for use in BEV application, Vehicle Power and Propulsion Conference (VPPC), IEEE, Lille, France, 2010.
- M. Otter, H. Elmqvist, S. E. Mattsson. Multidomain Modeling with Modelica. *Handbook of Dynamic System Modelling*, Chapman & Hall/CRC, chapter 36, pp. 36.1 - 36.27, 2007.

Modeling of an Automatic Transmission for the Evaluation of Test Procedures in a Virtual End-of-Line Test Bench

Jan Röper¹ Jörn Göres¹ Clemens Gühmann²

¹Daimler AG, Germany {jan.roeper, joern.goeres}@daimler.com

²Chair of Electronic Measurement and Diagnostic Technology, Technische Universität Berlin, Germany
clemens.guehmann@tu-berlin.de

Abstract

End-of-line tests for automatic transmissions are mandatory to ensure quality and safety. The interaction of unit under test, test bench and test automation leads to a high complexity in the development of test automation and test procedures. Validation of test automation and test procedures requires access to the test bench and the unit under test, both of which are only available close to startup of production. Therefore, virtualization of test bench and unit under test can be used to ease the bottleneck.

Virtualization is a common tool in the development of electronic control units for automotive applications using SIL and HIL technologies. The properties of simulation models for a virtual end-of-line test bench differ from those for classical SIL and HIL environments. In this paper, an automatic transmission model suitable for a virtual end-of-line test bench is presented. The required characteristics of the multiple-disk clutch friction model are discussed in detail. Hydraulics are modeled using a Moore machine to enable simulation of the pressure build-up characteristics during shift operation. With the resulting model, the influence of the key parameter of a test procedure actuating an overlapping gearshift is investigated in a virtual test system.

Keywords: automatic transmission, modeling, virtual test bench, HIL, SIL, end-of-line, friction, hydraulics, disk clutch

1 Introduction

Automatic transmissions (AT) are tested for quality and safety at the end-of-line (EOL). An example for a test regarding quality is the shifting of gears. The testing of the park break is an example for a safety related test. The EOL test system consists of a test bench, the unit under test (UUT) and a test automation. During the EOL test, test bench and UUT are stimulated by the test automation executing test procedures. The results of the tests are compared to limits and documented in a database for

statistical evaluation. Based on the test results, the AT is cleared or sent to rework.

When setting up a new test system, validation of the test automation as well as the test procedures is necessary to ensure smooth operation. Test bench and UUT are only available close to startup of production. Therefore, a virtual test bench is used as a substitute. A virtual test bench requires a model of the AT and the test bench. The control functions of the test bench programmable logic controller (PLC) have to be simulated as well. This also applies to the ECU functions used during the EOL test. For the testing of the automation and its test procedures, the simulation has to be connected to the test automation via the communication interfaces used in the real test bench. Figure 1 shows the components of the real test bench. All components except the test automation are simulated in the virtual test bench.

The main application of AT models in literature is the evaluation of system dynamics for controller design (Runde, 1984). The setup described above can be compared to HIL setups used for the evaluation of ECU functions. Isermann describes the history of HIL systems and shows its use for the development of engine control functions (Isermann et al., 1999). HIL simulation is also applied to PLC program testing for manufacturing equipment (Tomaszunas, 1998; Röck, 2007). However, the execution of ECU functions on none-ECU hardware is more characteristic for SIL setups (Chrisofakis et al., 2011). The virtual test bench is a composition of these virtualization techniques. The coupling of ECU and AT models is carried out in a customized third party SIL environment (Brückmann et al., 2009).

The customized SIL environment used for the implementation of the virtual test bench allows the utilization of real communication interfaces and a real-time simulation mode on a none real-time operating system. The test automation is insensitive to minor timing deviations. However, latency and jitter of the test system are affected by the real-time precision. The effects of latency and jitter on the test automation are discussed in (Röper et al., 2014) and have to be considered when working with the virtual test bench. The models of test bench and AT are

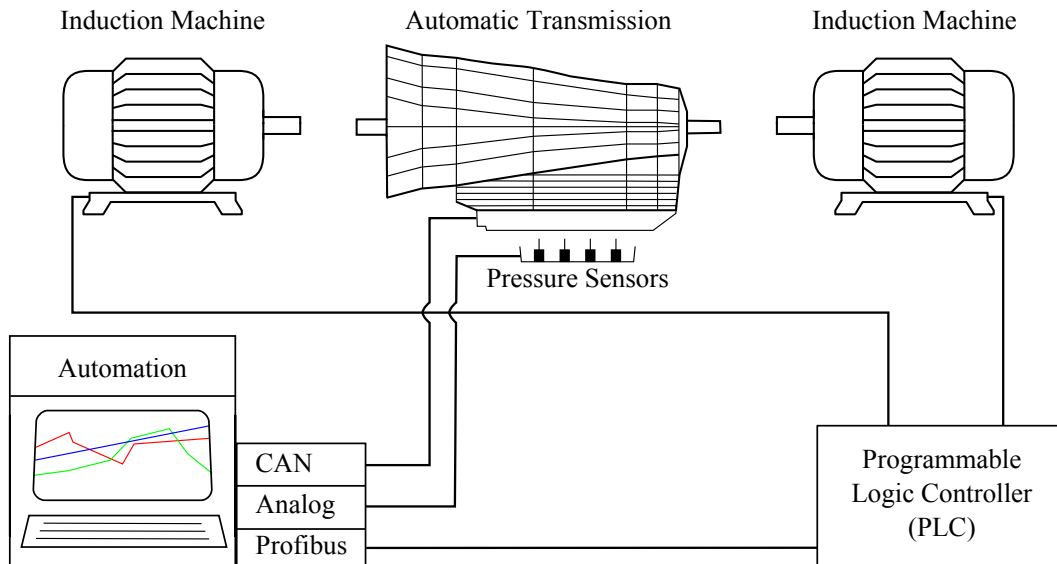


Figure 1. EOL test system

transferred from the Modelica[®] tool to the SIL as a functional mock-up unit. Simulation is carried out using a fixed step solver.

A similar setup uses a classical HIL as a base (Kuebler et al., 2012). No statements are made about the simulation of ECU functions. In the already published HIL-based implementation of a virtual test bench, the communication channels are merged to a single Profibus connection neglecting the influence of communication in the real setup.

In this paper, a model of an AT is presented, which fits the specific requirements of the EOL test. When creating a model for the virtual test bench in an early product design stage of the AT, there is a lack of technical data. Parameters which are unknown at this stage are estimated or reused from earlier projects. Such a model enables the test of the automation software as well as the development of test procedures and their pre-parameterization. When transferring test procedures developed in the virtual environment to a real test bench, the parameters have to be verified and possibly readjusted.

The focus in this paper is on the modeling of the characteristics of the pressure build-up in hydraulic clutch actuators as well as on close to zero slip in clutches. The focus results from the tests performed at the EOL: Pressure build-up characteristics have a strong impact on the synchronization during gear shift. After a successful gear shift, the transmission ratio is closely monitored to verify the ability of the clutch to transmit a specified torque. During the EOL test, the induction machine connected to the output shaft is set to speed control, while at the input shaft a constant torque is applied by a second induction machine. The transmission ratio R_T is calculated by:

$$R_T = \frac{\omega_{in}}{\omega_{out}}, \quad (1)$$

with the angular velocity at the input shaft ω_{in} and the angular velocity at the output shaft ω_{out} . A slip in any of the clutches leads to a decrease of the expected speed at the input shaft. In addition, the resulting ratio depends on the speed level which varies during the test. Therefore, the ratio is variable, even if the slip is constant. The following example demonstrates the effect: With an expected ratio of $R_T = 1$, a constant overall slip of $\omega_{slip} = 2 \text{ rad/s}$ would lead to a measured ratio $R_T = 0.996$ at the set speed $\omega_{out} = 500 \text{ rad/s}$, while the same slip would lead to a ratio $R_T = 0.98$ at a set speed of $\omega_{out} = 100 \text{ rad/s}$. The variability of the ratio R_T increases the demand for a model with low slip, since an adjustment of the speed levels may lead to the violation of a limit.

2 Modeling and Verification

The virtual test bench model consists of an AT model as well as models of the induction machines and their control. The modeling depth in each submodel is chosen depending on its significance for the EOL test. For example, the oil pumps of the AT are modeled with the states active and inactive, whereas the oil flow, depending on the input speed of the pumps, is not considered. Detailed submodels of the planetary gears, clutches and hydraulic actuators ensure a proper resemblance of the AT characteristics. Below, the detailed submodels of hydraulic actuators and clutch friction are presented. Both hydraulic actuators and clutches are nonlinear systems which require special treatment for real-time simulation. The verification of the test bench model is carried out with regard to its ability to simulate the sticking of closed clutches.

2.1 Modeling of Hydraulic Actuator

Prior to the modeling of pressure build-up in a hydraulic actuator, a detailed analysis of its design is carried out. Figure 2 shows a schematic diagram of a hydraulic actuator. The components of interest are the plunger I, the return spring II and the disks III. In addition, the characteristic pressure curve of the clutch actuator during activation and deactivation is shown as well.

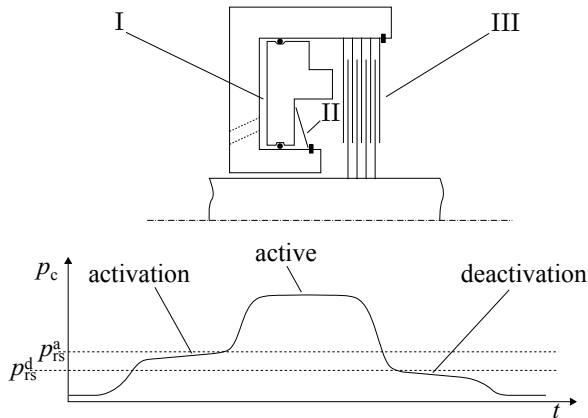


Figure 2. Clutch actuator and pressure characteristics

The clutch is activated by opening a valve that fills the hydraulic cylinder, which leads to the first pressure build-up in the curve. When passing the gap between plunger and disks, the pressure depends on the return spring characteristics. In AT clutches, disk springs are used to provide the return force, which leads to the first plateau at p_{rs}^a in the curve. Friction counteracts the plunger movement and adds to the return spring force. The second pressure build-up results from the contact force between plunger and clutch disks. The return spring characteristics also dominate the pressure curve when deactivating the clutch. The plateau during deactivation is on a lower pressure level p_{rs}^d , since friction forces act against the spring force when the plunger moves in its starting position. Both of the plateaus feature a small slope. The level of the plateaus as well as the slope is measured during the EOL test to ensure the assembly of the correct return spring type.

A physical model of the assembly presented above would require calculation of contact forces between plunger and clutch disks. This would result in a stiff system and therefore slow simulation due to the necessity of small integrator step sizes (Press et al., 2007). Therefore, an alternative approach is made to implement a real-time capable model. The model includes the electronic control valve which is driven by the ECU. In the model, the target pressure p_{set} of the actuator is calculated by a lookup table. The pressure build-up is dissected into states which can be represented by the Moore machine shown in figure 3 which is implemented using the Modelica[®] State Graph library.

The Moore machine gives out different levels of pres-

sure. State S_1 represents the inactive state, where a base pressure p_b ensuring deaeration is provided. State S_2 represents the plateau during activation of the clutch. In this state, the pressure $p_a(s)$ is rising with a specified slope. State S_3 represents the fully activated actuator where the target pressure p_{set} is reached, while state S_4 is the equivalent to state S_2 with the pressure $p_d(s)$ when deactivating the clutch.

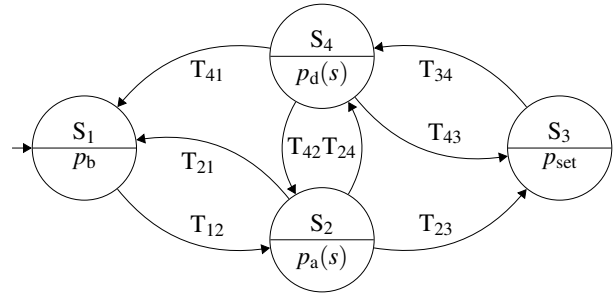


Figure 3. Moore machine of clutch actuator

The transition conditions are summarized by table 1. Fully activating and deactivating the clutch corresponds to a walkthrough of the states S_1 to S_4 in rising order followed by the state S_1 . Switching from state S_1 to state S_2 occurs when a signal is set by the ECU, which is big enough to generate a pressure that moves the plunger. State S_2 is kept active until the gap is covered. The gap is modeled by a linear differential equation:

$$s = \frac{1}{t_{gap}} dt, \quad (2)$$

with position variable s ranging from 0 to 1. The time needed for covering the gap is provided as a parameter t_{gap} . The height of the pressure rise during activation of state S_2 is provided by the parameter Δp_{rs}^a . State S_3 becomes active as soon as the condition $s \geq 1$ is satisfied and stays active until the pressure command p_{set} drops as specified in table 1.

Table 1. Transition conditions for Moore machine

	p_{set}	s
T_{12}	$\geq p_{rs}^a - \Delta p_{rs}^a$	
T_{21}	$< p_b$	≤ 0
T_{23}	$\geq p_{rs}^a$	≥ 1
T_{24}	$< p_{rs}^d$	> 0
T_{34}	$< p_{rs}^d$	
T_{41}	$< p_{rs}^d$	≤ 0
T_{42}	$\geq p_{rs}^a$	> 0
T_{43}	$\geq p_{rs}^a$	≥ 1

The signal generated by the Moore machine is a succession of straight segments with sharp angles. Both pressure build-ups occur instantaneously, which is not conform to the pressure build-up in the real system. The

pressure build-up can be modeled by the sum of two parallel PT2 element outputs y_{PT2}^1 and y_{PT2}^2 , whose parameters are tuned using measurements from a component test bench or simulation results from a physical model. The first PT2 element representing the pressure build-up before covering the gap is fully active when $s = 0$ is satisfied, while the pressure build-up after covering the gap is represented by the second PT2 element when $s = 1$ holds. While covering the gap, blend factors K_{B1} and K_{B2} proportional to s are used to calculate the sum of the PT2 elements:

$$K_{B1} = s, \quad (3)$$

$$K_{B2} = 1 - s, \quad (4)$$

$$p_s = y_{PT2}^1 K_{B1} + y_{PT2}^2 K_{B2}. \quad (5)$$

Figure 4 a) shows a measured pressure curve p_m and a simulated pressure curve p_s , both scaled by the factor $1/p_0$. The characteristic curves show a good agreement. Oscillations in the measured signal resulting from the pressure control are absent in the simulated signal, since the pressure control is not modeled. At $t = 0.4$ s the simulated pressure curve deviates from the measurement. The simplified model is not able to fully represent the physical effects when the contact between plunger and disks is established, which has an influence on the synchronization. The influence of the height of p_{set} on the pressure characteristics is neglected as well. Figure 4 b) shows the position variable s which rises linear during activation.

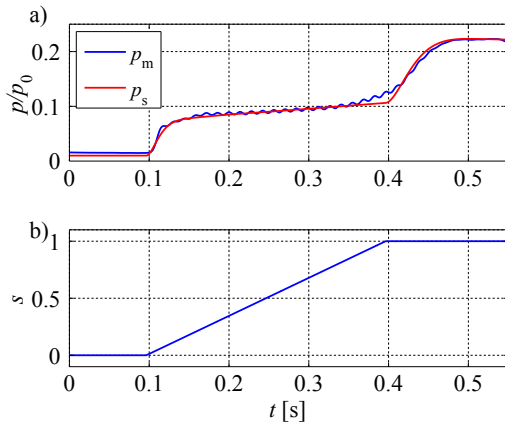


Figure 4. Clutch activation a) pressure and b) position

The utilization of the variable s enables realistic simulation of complex shift operations, for instance pre-fill of the clutch actuator or termination of a shift operation before synchronization. Figure 5 shows pressure, position variable and active state resulting from a normal shift I, an activation of the clutch from a residual pressure II (shift operation starts in state S_4 instead of S_1) and a prematurely terminated shift operation III. When activating the clutch from a residual pressure, the state

S_2 is held for a shorter time, since the position variable s is greater than zero when the pressure command is issued. The same effect occurs when terminating a shift operation. In this case, the pressure drop in state S_4 is taking less time than in the case of a normal shift operation.

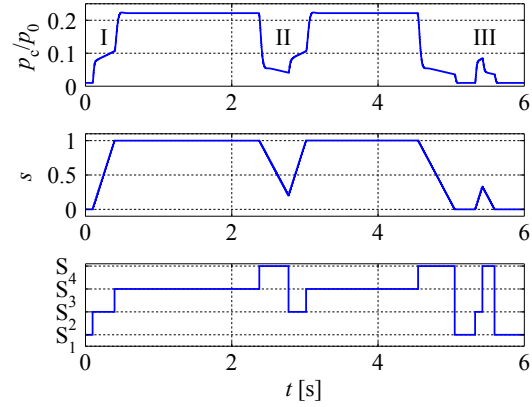


Figure 5. Shift operation scenarios I-III

2.2 Modeling of Multi Disk Clutch Friction

The hydraulic actuator model described in the preceding section enables the calculation of the normal force within the clutch. The normal force F_N and the clutch friction coefficient μ yield the ability of the clutch to transmit torque. The friction coefficient μ depends on the properties of the friction system, resulting from the tribological system of clutch disks and lubricant, as well as the relative speed v_{rel} of the shafts. The speed dependency of friction can be considered by expressing the coefficient of friction as a function of v_{rel} for a specific system. The clutch torque capacity T_{cap} defines the maximum stick and slip torque, which is determined by:

$$T_{cap} = N \frac{r_i + r_a}{2} \mu(v_{rel}) F_N, \quad (6)$$

with the inner disk radius r_i and the outer disk radius r_a as well as the number of disk couples N . Figure 6 shows the typical characteristics of the torque capacity in an AT disk clutch based on a measurement from (Mosbach, 2002). If $v_{rel} > 0$ is true, the transmitted torque T equals the torque capacity T_{cap} . In the case of zero relative velocity, T can take any value between zero and T_{cap} . In this case, T becomes a constraining torque resulting from the torque applied via the shafts. Mathematically, T is a set-valued function for $v_{rel} = 0$ (Lantos and Márton, 2011).

In the following section, the clutch friction system is replaced by a linear contact between a mass and a surface. The considerations made with this simplified system can be transferred to a rotational contact without modifications.

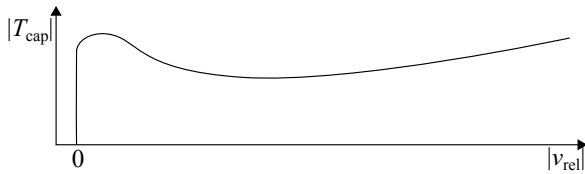


Figure 6. Torque capacity characteristic

There exist numerous friction models to determine the force transmitted when $v_{rel} = 0$. Mare gives a review of different friction models and categorizes them into three types depending on their ability to model contact dynamics (small tangential deformations in the contacting surfaces without sliding) and depending on whether the system dynamics properties are parameters of the friction model (Mare, 2012). The three types are:

1. *static and mass free*,
2. *dynamic and mass free*,
3. *mass integrated*.

In *static and mass free* models the set-valued function at $v_{rel} = 0$ is replaced by a function with a steep rise. Therefore, this type of model can not be used when true sticking with $v_{rel} = 0$ is required. The deviation of v_{rel} depends on the steepness of the function, which is limited by the computational effort. A steep rise leads to a stiff system which makes small step sizes necessary. An implementation of the type 1 model known as the *classic friction model* estimates the friction force F_{fric} with:

$$F_{fric} = \tanh(v_{rel}/v_0)\mu(v_{rel})F_N. \quad (7)$$

In *dynamic and mass free* models a differential equation is used instead of the algebraic function in type 1 models. Type 2 models are able to represent true sticking and are often based on physical effects in the contact. Typical examples are the *Bristle* and the *Reset-Integrator model* (Haessig and Friedland, 1991). While the *Bristle model* approximates the contact force by multiple elastic contacts, the *Reset-Integrator model* uses a single elastic contact which leads to faster computation. The parameterization of type 2 models requires knowledge of the underlying physical effects which is not available when creating models in an early product development stage.

The *mass integrated* type 3 models employ the external applied force which equals the friction force when sticking. They are simple for a single contact but lead to complex systems when multiple contacts are connected. The implementation of a reusable type 3 friction model for the simulation of an AT is not feasible, since the inertia properties and the structure of the AT are part of the type 3 model. The *Karnopp model* is the earliest implementation of a type 3 model (Karnopp, 1985). The effort for a multi contact implementation of the *Karnopp model* is shown by Deur for an AT model (Deur et al.,

2003). An additional submodel for the calculation of the torque at $v_{rel} = 0$ is implemented.

The Modelica[®] library supplies a clutch model which exploits event handling strategies (Otter et al., 1999). A classification using the three types from above is not possible. Characteristics of the event based friction model are true sticking and mass free. The model sets the relative acceleration a_{rel} to zero when a change of sign in the velocity is detected. The equation $a_{rel} = 0$ stays active until the force capacity of the contact is exceeded. This model satisfies the demand for small relative velocity in sticking mode, when using a variable step solver with an event searching algorithm. In the case of a sign change in v_{rel} , an event is triggered which stops the integrator. The point of time of the sign change is determined by an algorithm and the integrator is restarted with an adjusted set of equations.

When using a fixed step solver, as required for real-time calculations, the integrator is restarted without searching for the time of the sign change. Therefore, the residual relative velocity in sticking mode depends on the offset between the event and the integrator step after the event. The maximum relative velocity while sticking depends on the step size of the solver and the relative acceleration. The maximum error results from a sign change that takes place right after the integrator evaluated the equations. The time that passes until the event is detected is approximately equal to the integrator step size. The relative velocity resulting from the event based approach is constant once the sticking mode is established, contrary to the relative velocity acquired when using the *classical model*. Neither the classical nor the event based approach satisfy the requirements stated above.

A friction model that suits the requirements of the virtual test bench is the *kpki model* (Bai et al., 2013), which uses a structure resembling a limited PI controller. The original implementation does not allow modeling of a peak friction value at zero velocity. Figure 7 shows a novel, extended implementation of the *kpki model* supporting peak friction. The approach is based on an altered version of the limited PI controller provided by the Modelica[®] library.

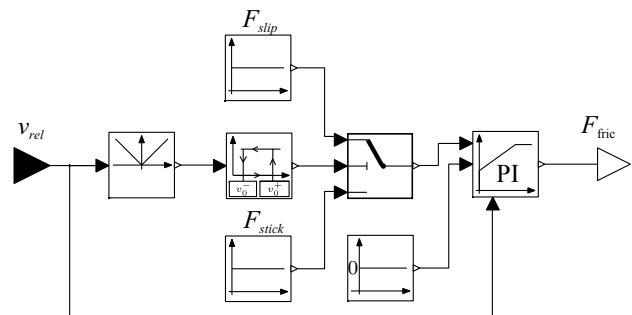


Figure 7. Extended *kpki model*

The PI controller is based on the work presented in

(Aström and Hägglund, 1995), where the limit in the controller is set as a parameter. Extending the model to the application of variable limits enables the support of peak friction at zero velocity. The limit is adjusted depending on the relative speed. Since the controller only reacts if $v_{rel} = 0$ is violated, a range representing zero velocity has to be defined. This range can be compared to the approach in the *Karnopp model*, where the parameter DV is used for this purpose. The model shown in figure 7 contains a hysteresis element defining two speed levels v_0^+ and v_0^- for the adjustment of the controller limit. This is necessary for a clean switchover to the stuck mode. If the relative velocity decreases and the switch to the peak friction level is made to early, the controller sets a high friction force to reach zero slip. Since the friction force in the physical system results from the external force, a rising friction force just before reaching zero speed is implausible. Figure 8 shows the curve of the simplified force characteristics implemented with a hysteresis element.

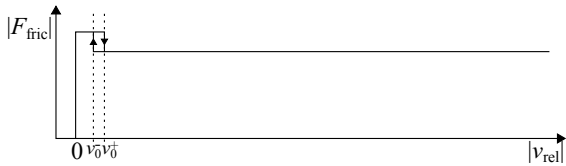


Figure 8. Friction force levels

A standard setup for the evaluation of friction models is shown in figure 9 (Haessig and Friedland, 1991). It consists of a mass with a friction contact to a surface, which is connected to a spring. The system is stimulated by applying a constant speed to the free end of the spring.

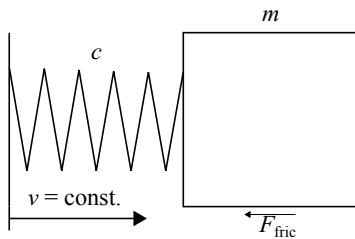


Figure 9. Friction experiment

The compression of the spring leads to a force F_c on the mass. The spring force rises until the peak value of the friction characteristic curve is exceeded ($F_{stick} = 0.25N$), resulting in an acceleration of the mass against the remaining friction force ($F_{slip} = 0.2N$). The movement of the mass relieves the spring until the damping of the friction brings it to a stop. The process is repeated in the same fashion as long as the spring end is moved. Figure 10 a) shows the resulting spring and contact force for the extended *kpki model* and figure 10 b) for the *classical model*. Both models are supplied with a friction characteristic with a peak at zero slip. The *classical model* does not have the ability to differ between rising and falling

speed, which leads to the peaks at the end of the sliding phase. The maximum relative speed during sticking is $5e-4m/s$ for the *classical model*, while the extended *kpki model* shows a maximum relative speed of $6.7e-7m/s$.

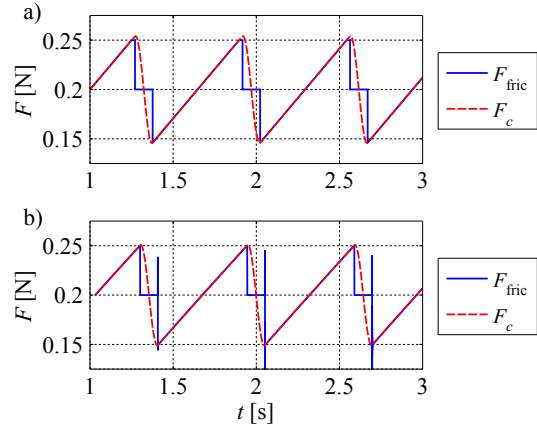


Figure 10. Friction experiment result a) *kpki model* b) *classical model*

The parameters of the model include gain K_{PI} and time constant T_i of the PI controller, as well as the velocities v_0^+ and v_0^- that define the switch points of the hysteresis element. When tuning the parameters of the extended *kpki model*, overshooting and oscillation of the friction force set by the PI controller have to be avoided. Therefore, an analysis of the poles of the system is carried out to fix the PI controller parameters. The time constant T_i is fixed at $T_i = 20h$ with the integrator step size $h = 0.0001s$ for the explicit Euler algorithm. The resulting numerical stability can be evaluated after the poles are set (Cellier and Kofman, 2006). The system boundary for the linear analysis includes the mass and the friction contact. Figure 11 shows the block diagram representing the model with the external force input U and the relative velocity output Y . The mass is represented by the transfer function $G_m(s) = K_m \frac{1}{s}$ with $K_m = \frac{1}{m}$, while the PI controller is represented by the transfer function $G_{PI}(s) = K_{PI}(1 + \frac{1}{T_i s})$, provided that the limiter is not active.

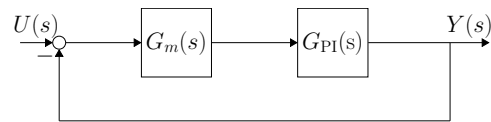


Figure 11. Block diagram

With the block diagram from figure 11 the transfer function of the system can be calculated to:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K_m K_{PI} s + K_m K_{PI} \frac{1}{T_i}}{s^2 + K_m K_{PI} s + K_m K_{PI} \frac{1}{T_i}} \quad (8)$$

A dynamic system is oscillatory if it holds at least two complex conjugate poles. The gain K_{PI} can be used to

move the poles to the x-axis of the left half pane of the pole-zero diagram to avoid oscillations. The poles can be calculated from the denominator of the fraction in equation 8 by completing the square:

$$-\frac{K_m K_{PI}}{2} \pm \sqrt{\left(\frac{K_m K_{PI}}{2}\right)^2 - K_m K_{PI} \frac{1}{T_i}}. \quad (9)$$

The poles are located on the x-axis of the left half pane if the square root term is none negative. When setting the parameters $m = 0.1 \text{ kg}$ and $c = 100 \text{ N/m}$ (Haessig and Friedland, 1991), a value of $K_{PI} > 200$ leads to a positive square root term and therefore to a dynamic system without oscillation. The best results regarding approximation of true sticking are obtained with $K_{PI} = 600$. The product of the resulting poles and the step width satisfy the stability domain of the explicit Euler algorithm.

After setting the parameters T_i and K_{PI} , the parameter v_0^+ can be calibrated by a simple experiment. Subjecting the mass to an external force leads to a relative velocity in the contact which is counteracted by the PI controller. The parameter v_0^+ has to be higher than the velocity induced by an external force step to the peak force level from figure 8. Setting a smaller value of v_0^+ triggers the lower friction level intended for the sliding mode. The same applies to an abrupt reduction of the external force to zero during sliding. A parameter v_0^- with the same value as v_0^+ would lead to a premature jump to the stick friction level. The effect of a premature jump to the stick friction level is visible in the simulation results from the *classical model* in figure 10 b). Therefore, a low value for v_0^- is desirable. If the value for v_0^- is chosen too small, the controller will never reach the stick mode after sliding. Figure 12 illustrates the performance of the model when stimulated by step signals. The relative speed resulting from a step to the maximum friction force (blue curve) stays within the boundary set by v_0^+ , while the relative speed resulting from a step to zero force (red curve) does not trigger the peak friction force after passing v_0^- . Both curves show a discontinuity, which results from the limiting within the PI controller (magnified area).

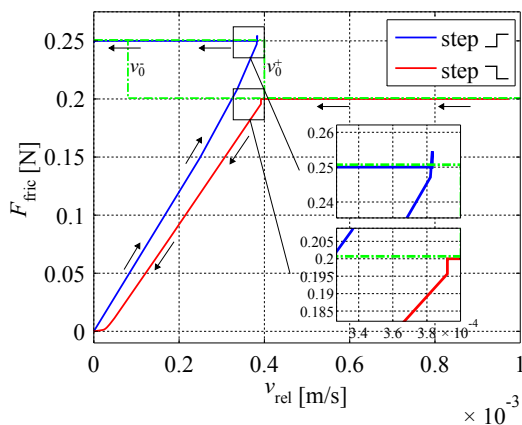


Figure 12. Calibration of v_0^+ and v_0^-

The model developed for the linear contact example can be transferred to a clutch model. When setting the parameters in a model with multiple clutches, the closed clutches which are not in the focus of the analysis at the moment can be replaced by ideal, slip-free connections. Modeling of drag torque in the clutch can be achieved by applying a small normal force even if the hydraulic actuator is inactive.

2.3 Verification

The suitability of the model for the execution of test procedures with regard to low slip in closed clutches has to be verified as well. The verification is carried out by measuring the transmission ratio while executing a test procedure on the virtual test bench. Table 2 shows the results of the verification and the design values R_T of the transmission ratio presented by (Dörr et al., 2014). The transmission ratio is evaluated in steady state $R_{T,\text{sim}}^{\text{stat}}$ and dynamic state $R_{T,\text{sim}}^{\text{dyn}}$. A representative steady state occurs after a successful gear shift. For evaluation of the dynamic state the worst case scenario was chosen: During acoustic measurement the load torque changes its sign, which demands rapid adjustment of the friction force. The transmission ratio is exactly emulated in steady state, while the deviations are small in dynamic state. The biggest deviation from the design value can be observed in the reverse gear with $\Delta R_{\text{sim}}^{\text{dyn}} = 0.019$, which is sufficiently small.

Table 2. Verification with test procedure

Gang	R_T	$R_{T,\text{sim}}^{\text{stat}}$	$R_{T,\text{sim}}^{\text{dyn}}$
1	5.503	5.503	5.496
2	3.333	3.333	3.330
3	2.315	2.315	2.311
4	1.661	1.661	1.658
5	1.211	1.211	1.203
6	1.000	1.000	0.996
7	0.865	0.865	0.863
8	0.717	0.717	0.715
9	0.601	0.601	0.600
R	-4.932	-4.932	-4.913

3 Simulation and Experiment

With the model presented above, a test procedure executing an overlapping gearshift is developed. The overlapping gearshift consists of a sequence of five steps illustrated by figure 13. After initialization of the test bench and the UUT, the signal for the activation of clutch C2 is sent to the AT. After an expiration of a sleep timer, clutch C1 is deactivated. Activating clutch C2 before deactivating clutch C1 is necessary due to the pressure build-up

characteristics discussed above. The choice of the sleep time parameter τ has a strong influence on the strain induced in the clutches during gearshift. A parameter study of τ is performed with the virtual test bench using the original automation in a safe environment. Choosing hazardous values for the parameter τ shows the consequences without damaging hardware. Measurements on a real test bench are carried out to confirm the characteristics identified by the simulation.

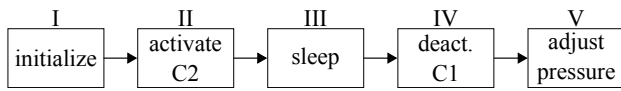


Figure 13. Overlapping gearshift sequence

3.1 Simulation

For the parameter study, the parameter τ is set with values from 150 to 400ms. Other parameters, like the pressure command for the clutch actuators, are constant during the simulation runs. Figure 14 shows the resulting pressures in the two clutches. The data is synchronized at the beginning of the pressure rise of clutch C2. The offset in time between the five resulting pressure curves for clutch C1 is clearly visible. The range chosen for the parameter τ covers gear shifts with no overlapping, as well as gear shifts with strong overlapping. In the pressure curves of clutch C2, the pressure adjustment in step V of the gearshift sequence is visible. The pressure adjustment is carried out with a fixed delay after synchronization of clutch C2.

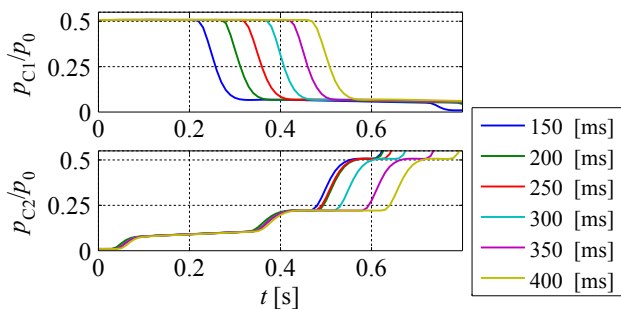


Figure 14. Simulated pressure during gearshift for different values of τ

Figure 15 shows the rotational speed of the input shaft ω_{in} , as well as the torque T of the input and output shaft. The influence of the sleep time on the synchronization is significant for $\tau \geq 300$ ms. In addition, the torque curves change as well. The torque on the input shaft is delayed for greater values of τ , while the characteristics of the torque on the output shaft change dramatically. An additional rise of torque before synchronization of clutch C2 is visible. For short sleep times τ , the synchronization is triggered by the rising pressure of clutch C2, while for longer sleep times the synchronization is triggered by the

falling pressure of clutch C1. In the case of longer sleep times, synchronization occurs on a higher pressure level leading to additional, undesirable strain within the transmission.

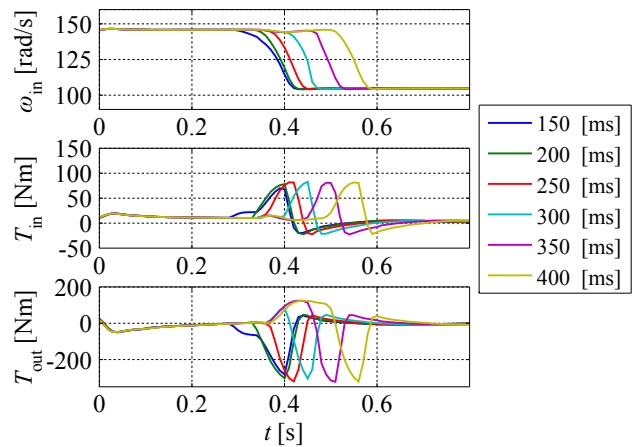


Figure 15. Simulated speed and torque during gearshift for different values of τ

The results from the parameter variation acquired in the virtual environment show that a maximum value of $\tau = 250$ ms leads to a clean overlapping gearshift. Longer sleep times lead to additional strain which can possibly damage clutch disks.

3.2 Experiment

To verify the simulation results, measurements with the test procedure presented above are carried out on a real test bench. The torque level predicted by the simulation does not exceed the torque tolerated by the test bench. Therefore, the measurements can be performed for all parameter settings used in the simulation. The AT used for the experiments may suffer increased clutch disk wear and is excluded from clearing.

Figure 16 shows the pressure curves measured during gearshift. The influence of the varied sleep time is clearly visible. The time interval between the falling pressures of clutch C1 is slightly irregular, which is caused by timing deviations in the test system. The signals show low scale oscillations which are not present in the simulation results. This is due to the simplifications in the model omitting pressure supply characteristics and clutch piston friction. The slope of the falling pressure in clutch C1 differs from the simulated clutch pressure slope, which can affect the time needed for synchronization.

Figure 17 shows the resulting curves for speed and torque. The synchronization of the measured input shaft speed ω_{in} is slower than the simulation result and shows an additional undershoot. The undershoot after synchronization is compensated in the simulation by the ideal speed control. The influence of the sleep time on the input shaft speed is significant for $\tau \geq 350$ ms,

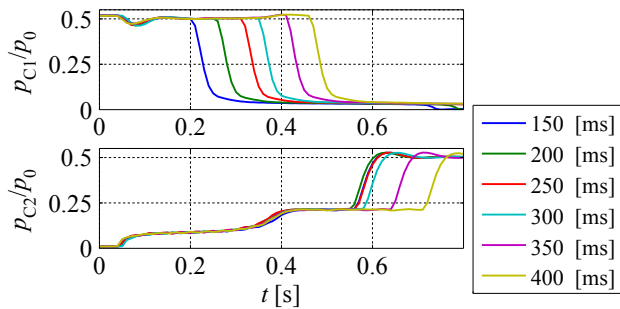


Figure 16. Measured pressure during gearshift for different values of τ

while the torque measurements are already affected for $\tau \geq 300$ ms. The maximum of the torque signal at the input shaft T_{in} shows a disagreement to the simulation. The inertias in the test bench model are only estimates, which leads to the deviation of the input shaft torque T_{in} . Characteristics of the measured and the predicted torque as well as the influence of the parameter τ are similar. The output shaft torque T_{out} shows a good resemblance, although the measured signal is smoother, which fits to the elongated synchronization.

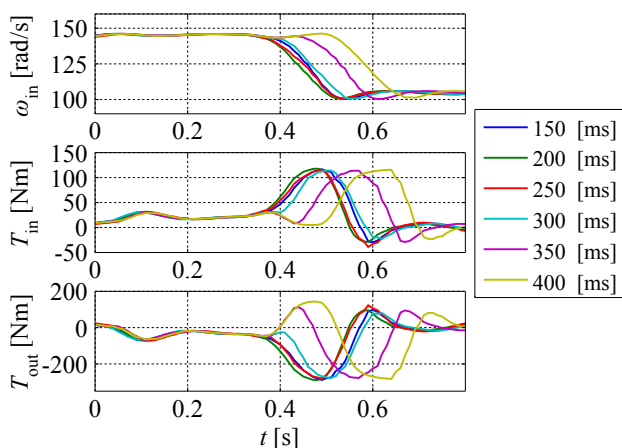


Figure 17. Measured speed and torque during gearshift for different values of τ

Altogether, the simulation shows a sufficient resemblance of the measured signals. The ability to estimate parameters for test procedures is confirmed. In the example shown above, a choice of the parameter $\tau = 250$ ms in the virtual environment is valid for the real test bench as well. This holds even though strong simplifications were made in the process of modeling.

4 Summary and Outlook

In this paper, an approach for the modeling of an AT in an early stage of product development is presented. In addition, real-time requirements resulting from the integration in a virtual test bench are met. Modeling of a hydraulic clutch actuator as well as modeling of friction

is discussed in detail. An extended implementation of the *kpci model* is applied to a synthetic example of a dynamic system and a method for the parameterization of the friction model is presented as well.

With the resulting virtual test system, the influence of the key parameter for an overlapping gearshift is investigated. The qualitative agreement of the results is confirmed by measurements on a real test bench. Possible improvements for a better quantitative agreement of the results obtained with the virtual test bench include:

- parameterization with speed dependent friction coefficient and exact test bench inertias,
- detailed modeling of controller properties of the induction machine,
- modeling of the effect of the valve target value in the hydraulic actuator.

References

- K. Aström and T. Hägglund. *PID controllers: theory, design and tuning*. International Society for Measurement and Control Seattle, WA, 1995.
- S. Bai, J. Maguire, and H. Peng. *Dynamic Analysis and Control System Design of Automatic Transmissions*. SAE International, 2013.
- H. Brückmann, J. Strenkert, U. Keller, B. Wiesner, and A. Junghanns. Model-based development of a dual-clutch transmission using rapid prototyping and sil. In *Getriebe in Fahrzeugen*, 2009.
- F. Cellier and E. Kofman. *Continuous system simulation*. Springer, 2006.
- E. Chrisofakis, A. Junghanns, C. Kehrer, and A. Rink. Simulation-based development of automotive control software with modelica. In *Proceedings 8th Modelica Conference*, 2011.
- J. Deur, J. Asgari, and D. Hrovat. Modeling of an automotive planetary gear set based on karnopp model for clutch friction. In *ASME 2003 International Mechanical Engineering Congress and Exposition*, pages 903–910. American Society of Mechanical Engineers, 2003.
- C. Dörr, H. Kalczynski, A. Rink, and M. Sommer. Nine-Speed Automatic Transmission 9G-Tronic by Mercedes-Benz. *ATZ worldwide eMagazines Edition.*, 01:20–25, 2014.
- D. Haessig and B. Friedland. On the modeling and simulation of friction. *Journal of Dynamic Systems, Measurement, and Control*, 113:1256–1261, 1991.
- R. Isermann, J. Schaffnit, and S. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7:643 – 653, 1999.

- D. Karnopp. Computer simulation of stick-slip friction in mechanical dynamic systems. *Journal of dynamic systems, measurement, and control*, 107:100–103, 1985.
- M. Kuebler, R. Ammann, and M. Wissbach. VIP, der virtuelle Getriebe-Endpruefstand. In *16. VDI Kongress: Berechnung, Simulation und Erprobung im Fahrzeugbau*, 2012.
- B. Lantos and L. Márton. *Nonlinear Control of Vehicles and Robots*. Springer-Verlag London, 2011.
- J. Mare. Friction modelling and simulation at system level: a practical view for the designer. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 226:728–741, 2012.
- C. Mosbach. *Das Reibungs- und Reibschwingverhalten nasslaufender Lamellenkupplungen*. PhD thesis, Technische Universität München, 2002.
- M. Otter, H. Elmqvist, and S. Mattsson. Hybrid modeling in modelica based on the synchronous data flow principle. In *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, 1999.
- W. Press, B. Flannery, S. Teukolsky, W. Vetterling, and T. Gould. *Numerical recipes, the art of scientific computing*. Cambridge University Press, 2007.
- S. Röck. *Echtzeitsimulation von Produktionsanlagen mit realen Steuerungselementen*. PhD thesis, Universität Stuttgart, 2007.
- J. Röper, J. Göres, and C. Gühmann. Analysis of timing and jitter in real and virtual test bench for automatic transmissions. In *Simulation and Testing for Automotive Electronics V*, 2014.
- J. Runde. *Modelling and Control of an Automatic Transmission*. PhD thesis, Purdue University, 1984.
- J. Tomaszunas. *Komponentenbasierte Maschinenmodellierung zur Echtzeit-Simulation für den Steuerungstest*. PhD thesis, Techn. Univ. München, 1998.

A New Fault Injection Method for Liquid Rocket Pressurization and Feed Systems

Zhu Mingqing¹ Xie Gang¹ Shao Jintao² Chen Liping¹ Zhou Fanli²

¹CAD Centre, Huazhong University of Science and Technology, Wuhan, China, 430074

²Suzhou Tongyuan Software&Control Tech. Co., Suzhou, China, 215123

{zhumq,xieg,shaojt,chenlp,zhoufl}@tongyuan.cc

Abstract

Fault simulation is an important method in the design of liquid rockets and fault injection is necessary for fault simulation. In this paper, we present a new fault injection method for liquid rocket pressurization and feed systems (PFS) without modifying the system structure. Firstly, we develop a physics-based model of pressurization and feed systems based on Modelica, which describes both nominal and faulty behaviors in a unified way. Then, we describe the new fault injection method, which uses the fault mode library and constructs the association between the Modelica model and the fault mode using customized Modelica annotation in MWorks®. We verify the new method by simulating several typical fault modes such as leakage and clogging. The results show that our method could be easily used to simulate various fault modes in liquid rocket pressurization and feed systems. Moreover, the new fault simulation process indeed plays a role in the system design. Our results could provide some reference for the ongoing research in fault detection and diagnoses.

Keywords: fault simulation; fault injection; Modelica/MWorks; pressurization; fault mode

1 Introduction

A suitable pressurization and feed system is important for a liquid rocket to transfer rocket propellants from the propellant tanks to the engine at certain flow rates and pressures (Partola, 2012). Mostly, the engine could not generate enough thrust to keep the rocket in orbit if the pressure inside the propellant tank is too low, which may lead to flight failure. Therefore, a diagnostic solution is needed to quickly identify the faults so that recovery actions can be taken or an abort procedure can be initiated before system safety is compromised (Daigle, 2011). Effective diagnoses require abundant historic data, which are traditionally acquired by executing many ground tests. Unfortunately, the costs of implementing ground tests are enormous and the process of physical tests is extremely dangerous. In addition, it is really hard or impossible to reappear some fault modes because of

equipment restrictions, let alone covering all possible flight conditions. Nowadays, with the development of computer science, fault simulation based on numerical methods is an ideal alternative to ground tests. Relying on a detailed model of system behaviors under nominal and faulty conditions (Daigle, 2011), numerical simulation has several advantages. Firstly, an engineer could build a mathematical PFS model according to its physical function and simulate its behaviors under all kinds of working conditions to verify the design of the system. Moreover, typical fault modes of a system could be manually injected into the nominal model to predict their effects on the system performance. In this way, we could accumulate abundant faulty knowledge of the system, which is important for ongoing research in fault diagnoses and design optimization.

Implementing faults in PFS is not new to the literature. For example, Gao Ming et constructed a filling system based on Modelica where fault simulation was processed by altering the models or resetting the parameters (Gao, 2009).

Wang Min et built a modular PFS based on Matlab where two typical faults were simulated by adding step signals to change the behavior of the system (Wang, 2010).

Fan Zhongze et simulated four kinds of faults by VC++ where the fault is expressed by the fault factor and fault trigger time (Fan, 2008).

Another approach for model-based fault simulation is used by F.L.J. van der Linden. Using instance modifiers as well as an inner-outer broadcasting method, the faults can be triggered in a central block (F.L.J. van der Linden, 2014). Though valuable in some aspects, this method is hard to handle a large number of fault modes.

Most of these works are examples in which faults in PFS are triggered and simulated. The simulation results can give some reference to the model-based diagnoses. However, all the implementations have to modify the original system model to inject fault modes, which is very fussy if there are many fault modes. In fact, engineers prefer a simpler and more convenient way to inject faults in system models.

In this paper, we develop a lumped-parameter dynamic model of PFS using Modelica on MWorks®, which is simple enough to allow for physics analyses and numerical simulations. More importantly, we propose a new method to describe the fault mode and inject the fault mode into the Modelica model without modifying the structure of the original system. Several typical fault modes are simulated to justify the validity of this method. We have investigated both the nominal regime and the effects of several primary faults, such as gas leakage in the gas pipe as well as clogging of the electric valve.

The paper is organized as follows. Section 2 gives an introduction to liquid rocket pressurization and feed system. Section 3 briefly describes the modeling of PFS considering both nominal and faulty behaviors. Section 4 presents the new method of model-based fault injection and the workflow of fault simulation. Section 5 verifies the approach with several fault simulation experiments. Conclusions and suggestions for future work are presented in section 6.

2 Overview of PFS

A simplified working diagram of the liquid rocket PFS is depicted in Figure 1.

The working process is outlined as follows. Initially, cold helium gas is injected into gas bottles 1 through valve 2 to obtain very high pressure on the ground. At the same time, the propellant tank is pressurized through ground valve 4. Once engine 17 starts, high pressure gas is released from the gas bottle, and finally enter the propellant tank. The mass flow rate of the gas is controlled by controller 12, which controls the opening of SV 6 or SV 8 by detecting the pressure in the propellant tank.

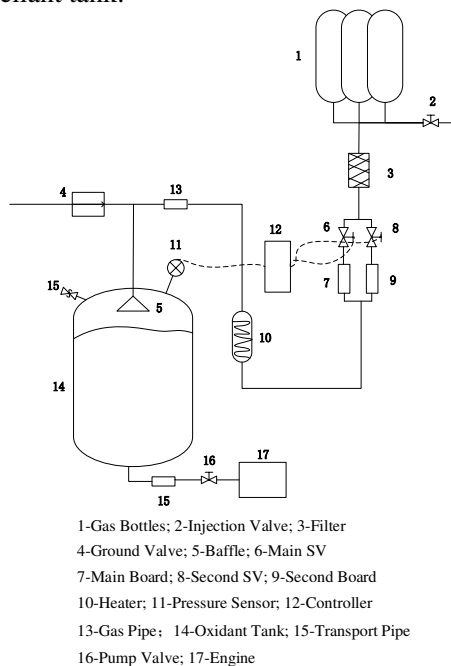


Figure 1 A Liquid Rocket PFS

3 Modeling PFS

A detailed system model is essential for fault simulation. Based on the above analysis, fluid dynamics and hydraulics, we could construct all the nominal mathematical models. To meet the requirements of both validity and computational performance, all the models are treated lumped-parameter. Both nominal and faulty behaviors are considered in the model.

Pressure Board

The purpose of the pressure board is to control the flow rate of the fluid from upwind. In fact, the board could be considered as an orifice. Based on the continuous equation, the isentropic relationship and the equation of the gas velocity, we could obtain the flow rate of the gas according to the pressure ratio (Esposito, 2000).

The mass flow rate is given by:

$$\dot{m} = A \cdot C_q \cdot C_m \frac{P_{up}}{\sqrt{T_{up}}} \quad (1)$$

Where C_q is the discharge coefficient, A is the restriction area, P_{up} is the upstream pressure, C_m is the mass flow parameter, and T_{up} is the upstream temperature.

The critical pressure ratio at which the flow switches from sonic to subsonic can be calculated using the equation:

$$P_{cr} = \left(\frac{2\gamma_s}{\gamma_s + 1} \right)^{\frac{1}{1-\gamma_s}} \quad (2)$$

Here, γ_s is the isentropic calorific factor.

When $(P_{dn} / P_{up} > P_{cr})$, the flow is subsonic and the flow parameter C_m is a function of the pressure ratio and depends on the gas properties. When $(P_{dn} / P_{up} < P_{cr})$, the flow is sonic, the mass flow parameter C_m is a constant and the mass flow rate only depends on the upstream temperature.

The nominal behavior of the board is described as in equation (1). To characterize the fault mode such as leakage, we introduce a leakage variable dm_flow and leakage coefficient K ($0 \leq K \leq 1$). The constrain equation is:

$$dm_flow = K \cdot \dot{m}_{g_in} \quad (3)$$

Here, K is the fault parameter of the board. $K=0$ implies the nominal condition and $K=1$ implies complete leakage.

Pipes (Gas and Liquid)

The flow condition of the fluid could be treated as laminar if the flow rate is relatively small and the mass flow rate changes linearly with the pressure drop (Esposito, 2000). The equation is given by:

$$\dot{m} = G \cdot dp \quad (4)$$

Similarly, we introduce a fault parameter *leak* to represent the leakage fault mode. The equation is given by:

$$dm_flow = leak \cdot \dot{m} \quad (5)$$

Here, *leak*=0 implies the nominal condition and *leak*=1 implies total leakage.

Gas Bottle

Gas Bottle is a closed volume filled with inert gas. Before the flight of the rocket, the gas bottle will be injected into enough gas to obtain high pressure. When the valve opens, the gas quickly expands into the downstream road. Since the gas flow is very fast, the heat transfer between the wall and the gas could be neglected and the temperature and pressure in the gas bottle would be given by (Esposito, 2000):

$$T = T_0 \left(\frac{p}{p_0} \right)^{\frac{\kappa-1}{\kappa}} \quad (6)$$

$$\frac{dp}{dt} = \frac{\kappa \cdot R \cdot T \cdot \dot{m}}{V} \quad (7)$$

Here, T_0 is the initial temperature in the gas bottle, p_0 is the initial pressure.

We introduce coefficient *leak* to represent the leakage behavior of the bottle. The equation is:

$$dm_flow = leak \cdot \dot{m} \quad (8)$$

Propellant Tank

The propellant tank is the most important component of a PFS. Tank pressurization is a very complicated physical process: the outer gas enters the tank and mixes with the initial gas in the tank, following the process of heat transfer and mass transfer. The tank wall is surrounded with the environment with all kinds of heat radiation and convective heat flow. Hence, it is rather difficult to predict the pressure in the tank. To determine the mathematical model, several assumptions are made as follows:

- The gas is undissolved with the liquid.
- The liquid is considered incompressible.
- The phenomena such as condensation and boiling are neglected.

Under low pressure, the gas could be treated as the ideal gas. The equation of state is given by:

$$pV = mRT \quad (9)$$

Ignore the kinetic energy of the entering gas. The mixed gas transfers heat with the tank wall and the liquid. The conversation equation is given by (Esposito, 2000):

$$\frac{dU_{gas}}{dt} = \sum \dot{m}_{in} \cdot c_p \cdot T_{in} - \sum (Q_1 + Q_2) - p_{gas} \cdot der(V_{gas}) \quad (10)$$

Here, U_{gas} is the internal energy of the gas, Q_1 is the heat flow between the gas and the liquid, Q_2 is the heat flow between the gas and the tank wall and V_{gas} is the volume of the gas.

Applying Newton Law, we obtain the heat transfer equation:

$$Q = hA\Delta T \quad (11)$$

Here, h is the coefficient of heat transfer, A is the heat transfer area and ΔT is the difference in temperature.

Since free convection is dominant in the tank interior, the average heat transfer is modeled based on the empirical Nusselt number correlations of the type $Nu = CX^n$ (Rohsenow, 1985).

$$h = \frac{k \cdot Nu}{L} \quad (12)$$

The characteristic length L in the above tanks different values for different heat transfer area.

The Modelica language for dynamic simulations is ideal for modeling PFS. Based on the equations described in this section, we could easily build all the corresponding Modelica model. Combining all the related components in Figure 1, we can develop a liquid rocket PFS as follows:

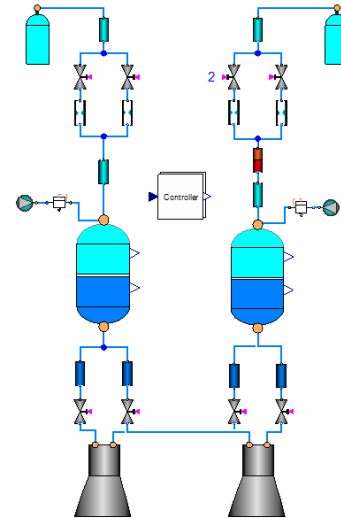


Figure 2 A System Model of PFS in MWorks Modeling View

4 Model-Based Fault Simulation

In the previous sections we have briefly discussed the nominal and faulty behaviors of the basic components. In this section, we describe how to associate the fault modes with the system model and inject the fault mode into a certain model correctly without changing the system topology.

4.1 Fault Mode Library

There are many kinds of fault modes in PFS. Traditionally, fault modes are stored in the literature as

depicted in Table 1. As is shown in Table 1, a single fault mode must be a faulty symptom of a certain product, which belongs to a subsystem. Many items are used to describe the fault mode and each kind of fault mode is tied with a fault parameter, as discussed in Section 3. However, all these items are written in the literature which could not be recognized by the computer. It is necessary to build a standard architect to transfer the literal fault mode into a quantitative fault mode file.

To get a unified description of all kinds of fault modes, we choose a XML file. XML is a hierarchy extensible language featured by self-describing, convenient data processing and easy understanding. It is suitable for the representation of the fault modes of a liquid rocket system. Moreover, it makes it convenient for us to generalize this method to other tools (not limited to MWorks) in the future. According to the requirements of the configuration information in Table 1, the XML file should include at least three contents: the product name, the fault mode description and the fault parameter. Another key element is the trigger time, since most of the fault modes are not triggered at the beginning. The logical hierarchy of the XML file is designed in Code 1.

Table 1 Literal Description of Some Fault Modes

Subsystem	Product	Fault Mode	Fault Parameter
PFS	GasBottle	Leakage	<i>leak</i>
	Board	Leakage	<i>K</i>
		Clogging	<i>leak</i>
	Valve	Always Open	<i>leak</i>
		Always Closed	<i>leak</i>
	GasPipe	Leakage	<i>leak</i>
LiquidPipe	Leakage	<i>leak</i>	

Code 1 A Sample of a Certain Fault Mode

```

</Product>
  <Product Type="gaspipe" Name
="gaspipe" >
    <FaultMode Name="Leakage">
      <Parameter Name="leak"
Value="0" Condition="time>10" />
    .....
  </Product>

```

Code 1 describes the leakage fault mode of a gas pipe according to Table 1. There is a default value for the fault parameter and trigger time, which can be modified before simulation. Moreover, the XML file can be extended to add some new items.

To better modify the XML file from GUI, we have also designed a XML editor on MFC (Petzold, 1998). The XML file can be read and modified by utilizing the DOM (document object module) technology, which is

a cross-platform and language-independent convention for representation and interacting objects in HTML, XHTML and XML documents.

Similarly, we could build other fault modes in the XML file just as in Code 1. By extending MWorks (Zhou, 2006), we develop an interface to read the XML file into the simulation environment. The structure of the FML (fault mode library) is depicted in Figure 3.

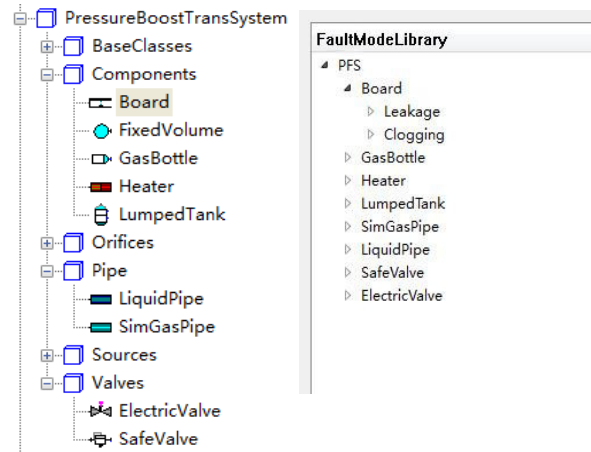


Figure 3 Hierarchical Structure of PFS Library and FML

4.2 Mapping Between Fault Mode and Modelica Model

From the above discussion, we know that faulty behaviors are stored in the Modelica file and fault modes are stored in the XML file. They are implicitly connected via the fault parameter. We need to construct a mechanism to make sure that the fault mode can be injected into the right object.

As the Modelica semantic describes, annotation is an attribute or property containing information associated with some element of a Modelica model and it will not affect the simulation of the model (Peter, 2010). According to the annotation syntax, we introduce a new annotation mapping for associating Modelica model with the fault mode in FML. Take the SimGasPipe leakage fault as an example:

Code 2 SimGasPipe Modelica Code

```

model SimGasPipe
annotation (__MWorks (FaultInfo (ModelName="gaspipe")));
extends
DynamicSystem.PressureBoostTransSystem.
BaseClasses.TwoPortsSysGas;
  Real leak = 0.1 "leakage coefficient"
  annotation
  (__MWorks (FaultInfo (FaultParameter)));
  parameter Real G (unit="kg/s/pa") = 1000;
equation
  m_flow = G * dp;
  dm_flow = m_flow * leak;

```

As we observe from the code above, the first kind of annotation defines the name of the model which must be consistent with the product type defined in the XML

file. The defined word *ModelName* is mapped with the string *Type* defined in XML file. The second kind of annotation shows that this variable is a fault parameter and can be changed by fault injection. The defined word *FaultParameter* matches with *Parameter* in the XML file. Both fault modes start with keyword *_MWorks* to make sure that it can be identified by MWorks.

4.3 Fault Injection

Fault injection is a key process in fault simulation. All faults could be injected into a system by changing the system input in some way (Genler, 1991). In the state-space framework, the process is described in the following way:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + Fp(t) \\ y(t) &= Cx(t) + Du(t) + q(t) \end{aligned} \quad (13)$$

Here $u(t)$ and $y(t)$ are the command value of the inputs and the measured value of the outputs, respectively, $p(t)$ and $q(t)$ are the fault vectors and F is the fault entry matrix. Vector p contains the actuator faults, disturbances and input sensor faults. Vector q contains the output sensor faults. The fault entry matrix F is assumed to be known and could be considered as a constant input to the whole system.

Based on the mapping relationship discussed above, we could make fault injection possible by changing the value of the fault parameter in the model. The so-called fault parameter has a meaning different from the definition in Modelica. It can be treated as an input signal to the system as discussed in equation (13). To be able to change the value of the fault parameter during the simulation, we should define it as a variable. Figure 4 helps us better understand the idea.

```

model ElectricValve
  annotation (__MWorks(FaultInfo(ModelName = "ElectricValve")));
  extends BaseClasses.TwoPortsSysGas;
  Real leak = 0 "leakage coefficient";
  annotation (__MWorks(FaultInfo(FaultParameter)));
  parameter Real G(final unit = "kg/(s.Pa)") = 1000;
  Modelica.Blocks.Interfaces.BooleanInput opening
    annotation (extent = [-20, 60; 20, 100], ...);
equation
  dm_flow = leak * m_flow;
  m_flow = if opening then G * dp else 0;
end ElectricValve;
    
```

leak=if time >10 then 0.8 else 0

	ParameterName	Value	TriggerCondition
FaultInfo	leak	0.8	time>10

Fault Injection

Figure 4 Fault Injection Operation

As can be seen from Figure 4, *leak* is the fault parameter defined in the model. The fault mode will be triggered at time 10s. The expression “leak= if time>10 then 0.8 else 0” depicted in Figure 4 is only a pseudo code for easy understanding of the injection mechanism. The platform MWorks will not generate such an equation when compiling the model but just replace the default fault parameter 0 with the new setting parameter 0.8.

4.4 Fault Simulation

To help the user better understand the new fault injection method, we give a fault simulation procedure as follows:

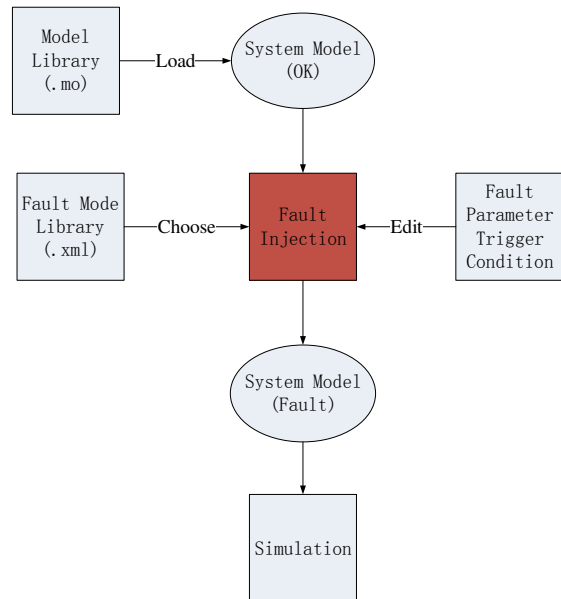


Figure 5 Fault Simulation Process

Firstly, we construct a system model from components in the PFS library. Secondly, we choose the fault mode that is to be simulated from the fault mode library. It is worth to note that there may exist more than one object of the same class in the system for a certain fault mode. For example, there are four SVs, all of which are classed as electric valve in Figure 1. We must make sure that the fault mode is associated with the right object. This problem is resolved by the new annotation mapping. Specifically, MWorks will associate all the related objects in the system model and let the end user choose which object to be injected. Thirdly, we need to specify the fault parameter and fault trigger time. An exception is the nominal situation where there is no need to handle this. Finally, we simulate the system model. At that time, the fault parameter in the model will be replaced by the value we specified if the fault trigger condition is satisfied.

5 Case Study

Based on the system depicted in Figure 2, we first conduct nominal condition simulation to demonstrate the validation of the system model. Then, we show to how to inject several typical fault modes were injected using the method described in Section 5.

5.1 Nominal Condition

Under nominal condition, the propellant pressure should maintain a relatively high level to ensure a safe flight. The simulation result is as follows:

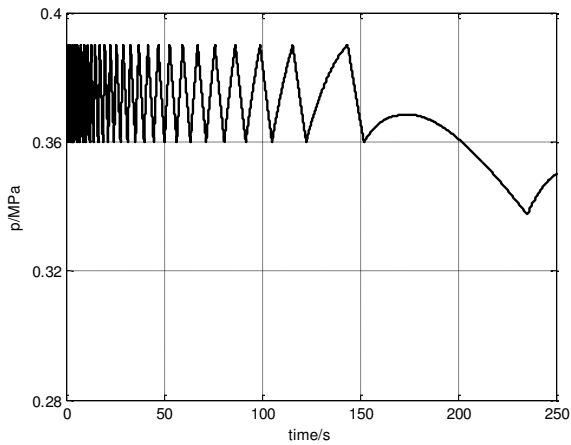


Figure 6 Pressure in the Tank@ Nominal Condition

As we can see from figure 6, the pressure is always within the setting pressure range of SV6 before 150s and SV8 is always closed. After 150s, the gas bottle could not supply enough gas to the propellant tank and the pressure experiences a little drop until 250s. The pressure drops under the minimal control pressure of SV8, causing SV8 opens to increase the pressure of the tank. It can be seen from Figure 6 that the pressure in the tank always keeps a relatively high level and satisfy the requirements of the flight. The system model is well suited for the simulation of PFS.

5.2 Fault Simulation-SV Always Closed

To overcome accidental faults in the flight of a rocket, redundant strategy is always adopted for an easily broken part or a critical part. As is done in Figure 1, two parallel SVs could ensure to some extent the safety of the rocket.

As seen from Figure 7, a fault mode is injected into SV6 at time 50s, which makes the valve always closed. The pressure in the tank drops sharply below the minimum control value of SV8 and it opens to ensure that the pressure keeps oscillating within its setting range. The simulation result shows that the fault injection method is successful and the redundant strategy is helpful in maintaining the tank pressure.

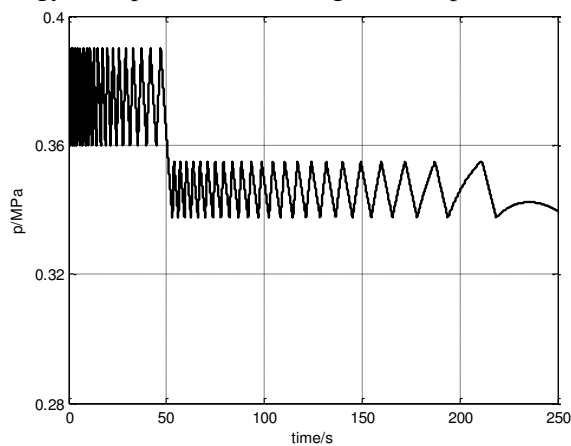


Figure 7 Tank Pressure @ SV6 Fault

5.3 Fault Simulation-Gas Pipe Leakage

Figure 8 shows the result of the tank pressure with a modified leak coefficient. The tank pressure could sustain if the leakage is 20 percent, whereas the pressure drops sharply if the leakage is more than 40 percent. The simulation result demonstrates that the fault injection method is successful and the system has a high level stability even under a fault mode (20% leak-age).

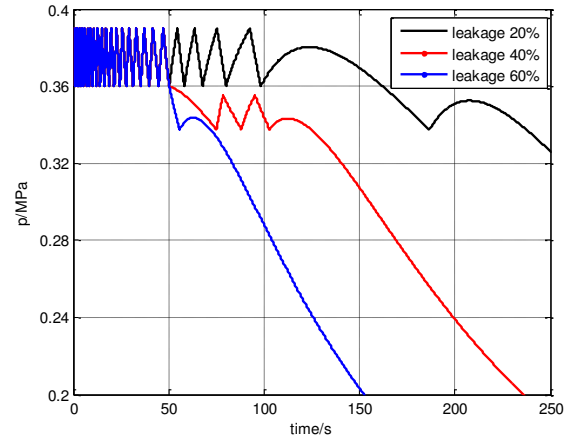


Figure 8 Tank Pressure @ Gas Pipe Leakage

6 Conclusions and Future Work

In this paper, we develop a physics-based model of pressurization and feed systems and analyze the validity of the model. Moreover, we introduce a new annotation mapping mechanism to associate the Modelica model with the fault mode, which lays a foundation for fault injection without modifying the structure of the system model. Standardization of different kinds of fault modes using a XML file is realized as well.

By adjusting the fault parameter of the gas pipe, we obtain simulation results that are consistent with the qualitative analysis, thereby justifying the validity of the new fault injection method. The new method is easy to understand and can support multiple fault modes injection is also supported as well.

Based on the above analysis, we conclude that fault simulation could help us better understand the system and explore the weakness of the system design in advance. The new fault injection method is simpler and more convenient than the traditional methods.

The idea of fault injection is quite general. It is not limited to PFS and can be applied to other domains by simply modifying the fault modes and system model. PFS is just one application and we plan to explore this method further in the future. For example, fault transmission and fault diagnose are on our agenda. We could simulate the process of fault transmission and do some fault diagnose analyses by batch simulation based on current method. Another future research is to combine current work with FMEA efficiently.

Acknowledgements

The paper is supported by The National High Technology Research and Development Program of China ("863"Program) (No. 2013AA041301)

References

- Partola I S. Design of liquid-propellant rocket engines. *Journal of Machinery Manufacture and Reliability*, 41(6):492-498, 2012.
- Daigle M, Foygel M, Smelyanskiy V. Model-based diagnostics for propellant loading systems. *Aerospace Conference, 2011 IEEE*. pp. 1-11, 2011.
- Gao Ming, Niaoqing HU, and Guojun QIN. Object-oriented Modeling and Fault Simulation of Propellant Filling System. *Machine Tool & Hydraulics*, 37(09):223-226, 2009.
- Wang Min, Hu Niaoqing, Qin Guojun. Fault Modeling and Simulation Analysis for LRE Test-bed Filling System. *System Simulation*, 22(11): 2672-2675, 2010.
- Fan Zhongze, Huang Minchao. Fault Simulation of Space Power System in the Operation Process. *Journal of National University of Defense Technology*, 30(02): 11-15, 2008.
- F.L.J. van der Linden. General fault triggering architecture to trigger model faults in Modelica using a standardized blockset. *Proceedings of the 10th International Modelica Conference, 2014*.
- Esposito A. *Fluid power with applications*. Prentice-Hall International, 2000.
- Rohsenow W M, Hartnett J P, Ganic E N. *Handbook of heat transfer fundamentals*. 1985.
- Petzold C. *Programming windows*. Pearson Education, 1998.
- Fan-Li Zhou, Li-Ping Chen, Yi-Zhong Wu, Jian-Wan Ding, Jian-Jun Zhao, Yun-Qing Zhang. MWorks: a Modern IDE for Modeling and Simulation of Multidomain Physical Systems Based on Modelica. *Proceedings of the 5th International Modelica Conference, Vol. 2: 725-731, 2006*.
- Peter Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons. 2010.
- Genler J. Analytical Redundancy Methods in Fault Detection and Isolation. *Preprints of IFAC/IMACS Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS'91*. 1991.

Automated Safety Analysis by Minimal Path Set Detection for Multi-Domain Object-Oriented Models

Christian Schallert

Institute of System Dynamics and Control, German Aerospace Centre (DLR), Christian.Schallert@dlr.de

Abstract

This paper describes, exemplifies and substantiates a method for detection of the minimal path set of any fault-tolerant technical system that is represented as a multi-domain object-oriented model. Thus, the method automatically performs a safety or reliability analysis of the system.

Keywords: safety analysis, reliability analysis, minimal path set, graph algorithms, modelling of failures, failure probability

1 Introduction

Safety and reliability are essential in transport aircraft design and operation, as well as other technical areas. Safety analyses are therefore an inherent part of the complex process of aircraft and on-board systems development. In systems development, multi-domain object-oriented modelling and simulation have now become the state-of-the-art.

This paper describes a method that integrates safety or reliability analysis with multi-domain object-oriented modelling. In essence, the method automatically detects the minimal path set of any fault-tolerant technical system. The method is based on the simulation of normal behaviour, degradation and failure of a system. Thus, modelling of failures is supplemented to component models from generic libraries, e.g. the Modelica Standard Library, that typically represent only normal, intact behaviour.

Other approaches to automated safety or reliability analysis based on multi-domain object-oriented modelling exist. A model-based diagnosis approach has been described by (Bunus, Lunde, 2008) that uses constraints (inequalities) instead of differential equations. It is particularly dedicated to diagnosing systems, i.e. detecting and isolating faults. Another approach described by (Papadopoulos *et al.*, 2001) performs semi-automatic fault-tree synthesis based on fault annotations included in the components of a system model.

The method described in this paper differs from the existing approaches, in so far that it uses differential-algebraic equations and modelling of failures. It thus permits the conducting of all other simulation studies that initially motivated the implementation of a model,

as well as it ensures a consistent safety analysis due to the modelling, not just annotating, of failures. The goal of the method is to improve the development process of fault-tolerant, safety-critical systems.

2 Modelling Approach

This section refers to the approach selected for the modelling of fault-tolerant systems and the additions necessary to enable automated safety analysis.

2.1 Modelling of Failures

The proposed minimal path set detection method requires that failure of a system can be simulated in addition to its normal behaviour. Thus, the modelling has to be supplemented by equations that reflect failures of system components and, if applicable, by operating logics that determine how a system reacts to the occurrence of component failures.

Model parameter values are changed in order to represent a failure. In doing so, the model equations remain the same (structure-invariant approach). Corresponding examples of aircraft on-board system models including component failures, e.g. electrical open circuit, mechanical disconnection or loss of hydraulic pressure, are provided in (Schallert, 2008, 2011, 2014). The proposed detection method activates component failures by directly accessing the relevant model parameters. Alternatively, a universal fault triggering network described by (van der Linden, 2014) can be used for activation of failures.

Provided that the preconditions (see subsection 3.1.2) are met, the detection method can be used also if the structure of the model equations is changed to represent failures. Such a structure-variant, multi-mode approach is described by (Elmqvist *et al.*, 2014).

Component failure rates λ_i are stored in each component model that includes failures. Since the λ_i values are used only for post-processing (see equation 2), they can be inserted also as custom annotations; a concept described by (Zimmer *et al.*, 2014).

2.2 Indication of System Status

Safety or reliability assessment requires the analyst to define criteria that indicate if a system operates normally or if it fails. Such criteria have to be

implemented in a system model, in order to compute a s_{sysOp} output signal that indicates system operation.

In case of a flight control surface actuation system, such as described in (Schallert, 2014), s_{sysOp} is computed by comparing the actual position or rate of the controlled surface with the command (model input). The capability of the system to follow commands is simulated by the minimal path set detection method for various combinations of intact and failed components. In doing so, s_{sysOp} is evaluated for correlation with the respective component states.

3 A Method for Minimal Path Set Detection

In this section a method is described that solves the problem of determining the failure probability of a system by detecting its minimal path set. The method is called DMP. It draws on a representation of the system model object structure as a graph and on simulation. Minimal path set analysis generally assumes that a system and its components are two-state, intact or failed, as explained in section 2.3 of (Biolini, 2007).

The DMP method is a state space simulation. The state space, in this context, denotes the set of all combinations of intact and failed components of a system to be examined for detection of its minimal path set. Evaluation of the system graph reduces the size of the state space and hence the number of simulations required.

3.1 Definitions and Preparations

3.1.1 Definitions

Definitions are provided of the terms used in the following for the DMP method:

Set. A defined collection of distinct objects, e.g. the components of a system.

Subset. A is a subset of B , $A \subseteq B$, if every object of A is also an object of B , e.g. $\{1, 2, 3\} \subseteq \{1, 2, 3\}$. If A is a subset of but unequal to B , then A is a proper subset of B , $A \subset B$, e.g. $\{1, 2\} \subset \{1, 2, 3\}$.

Superset. A is a superset of B , $A \supseteq B$, if every object of B is also an object of A , e.g. $\{1, 2, 3\} \supseteq \{1, 2, 3\}$. If A is a superset of but unequal to B , then A is a proper superset of B , $A \supset B$, e.g. $\{1, 2, 3\} \supset \{1, 2\}$.

Difference set. $A \setminus B$ denotes the set of elements that are members of A but not of B , e.g. $\{1, 2, 3\} \setminus \{2\} = \{1, 3\}$, or $\{1, 2, 3\} \setminus \{4\} = \{1, 2, 3\}$.

Component. A distinct element of a system. In this paper, components are also called nodes.

Combination. A set of intact components of a system.

Path. A set of intact components that causes a system to operate.

S-T path. A Source-to-Target path in a graph.

Path set. A set of paths of a system.

Minimal path. A path that cannot be reduced without causing system failure.

Minimal path set. The set of all minimal paths of a system.

Graph. A representation of a set of objects, e.g. the components (nodes) of a system, and of the connections between them.

Node. An object in a graph. Nodes are also called components in this paper.

Edge. A link that connects a pair of nodes in a graph.

Articulation. A node in a graph (or path) that, if removed, disconnects the graph (or path) into several subgraphs.

Subgraph. A part of a graph whose set of nodes and set of edges are subsets of those of the graph, the set of edges being restricted to the subset of nodes.

Density. The density d of a graph is generally, e.g. in (Diestel, 2010), defined by

$$d(N, E) = 2E/N(N - 1) \quad (1)$$

where N and E denote the numbers of nodes and edges of the graph, respectively.

Probability computation. The probability of system operation or failure is computed from the system's minimal path set in applying the reliabilities of its components. Let C_i denote the intact state of component i . Then, the probability of occurrence P of a minimal path MP is, see (Meyna, Pauli, 2003),

$$P(MP) = P(C_1 \wedge C_2 \wedge \dots) \quad \forall C_i \in MP$$

$$P(MP) = \prod_{C_i \in MP} P(C_i) = \prod_{C_i \in MP} R_i(t), \quad R_i = e^{-\lambda_i t} \quad (2)$$

with the component reliabilities R_i , failure rates λ_i and exposure time t . Exponentially distributed lifetimes are assumed. Other lifetime models, e.g. Weibull distribution, can be used as well. The probability of system operation $R_{\text{sys}}(t)$ is computed from the probabilities of the minimal paths by

$$R_{\text{sys}}(t) = P(MP_1 \vee \dots \vee MP_r)$$

$$= \sum_{j=1}^r P(MP_j) - \sum_{j=1}^{r-1} \sum_{k=j+1}^r P(MP_j \wedge MP_k) + \dots \quad (3)$$

$$+ (-1)^{r+1} \cdot P(MP_1 \wedge MP_2 \wedge \dots \wedge MP_r)$$

where r is the number of all minimal paths in the set. Equation 3 is evaluated for illustration at the end of subsection 3.2.2.

3.1.2 Properties of Minimal Paths and Requirements for Detection

This subsection explains the assumptions and requirements that apply to the minimal path set detection method DMP described in section 3.2:

1. The system behaves monotonously. This refers to a system that operates if all its components are intact and fails if all components fail. If the system

operates while not all components are intact, it continues operating if any further component becomes intact. Conversely, if the system has failed, it remains failed if any further component fails. This definition of monotony is common in safety analysis. For instance, it can be found in section 14.2 of (Meyna, Pauli, 2003).

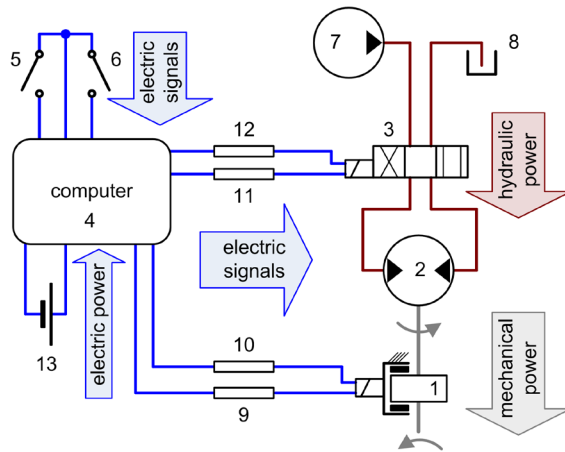
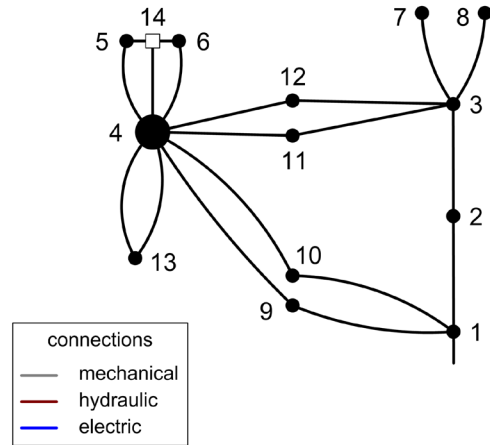


Figure 1. Exchange of power and signals across the edges in a coherent set of nodes

Definition 1. A set of nodes in a graph is coherent if any two nodes of the set are connected through a series of edges and through only those nodes that belong to the set.

Figure 2 shows coherent and incoherent sets of nodes (marked blue) for illustration. An S-T path, such as (c), is a special case of a coherent set of nodes.



- Every real world component is represented by one model object and by one node in a corresponding graph. No component is represented by two or more model objects or nodes.

DMP relies on a representation of the object structure of the system model as a graph. Nodes of the graph represent components, and edges the connections between components. The establishing of a graph is described in subsection 3.1.3. The properties of minimal paths, and in particular their situation in the graph, are explained in the following, which then proceeds to further requirements for DMP.

Depending on the system model and, if applicable, the marking of sources (S) and targets (T) in the corresponding graph, some S-T paths are minimal paths. This is true, for instance, for the electric network models shown in (Schallert, 2008, 2011), where also related detection methods are described. In general, however, what is known is only that a minimal path consists of one or more connected nodes.

This is explained by Figure 1 that depicts a part of an aircraft’s flight control surface actuation system model and its accompanying graph. The edges of the graph correspond to the interfaces that exchange power, material or signals among the components (nodes) of a system. This exchange among neighbored nodes enables a system to operate. No other nodes are situated between any of those nodes that exchange power, material or signals and hence belong to a minimal path. Thus, only a coherent set of nodes can be a minimal path. The following defines a coherent set of nodes:

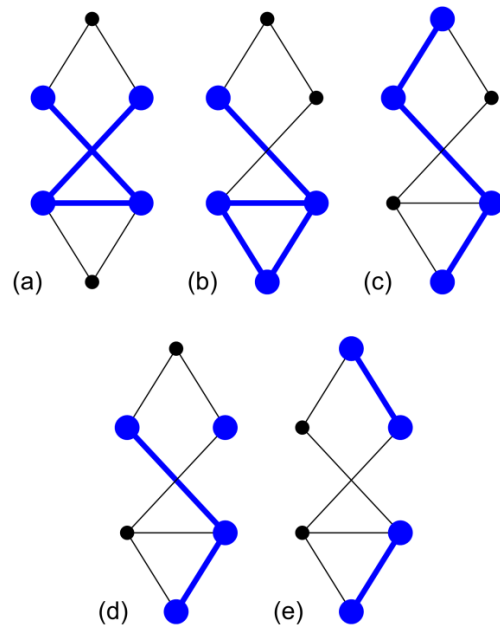


Figure 2. Coherent (a), (b), (c) and incoherent sets of nodes (d), (e) in a graph

Coherence (interconnection) of intact nodes in the system graph is a precondition for a minimal path. Removing a node from a minimal path interrupts the exchange of power, material or signals among the nodes of the minimal path. If no other minimal paths exist, the system fails. If the system operates with an incoherent set of intact nodes, nodes can be removed from the set, i.e. fail, without interrupting the exchange of power etc. Such a set of nodes is therefore a path but not a minimal path. Thus, the third assumption for method DMP is:

3. Only a coherent set of intact nodes in a system graph can be a minimal path.

Because not every coherent set of intact nodes is a minimal path, the system model is simulated to determine which ones are actually minimal paths.

3.1.3 Graph Representation of Multi-Domain Object-Oriented Models

A graph is defined by its adjacency list (array AL). In AL , each row corresponds to a node of the graph. The neighbours of a node are stored in the respective rows of AL , as will be illustrated. If more than one connection exists between two components of a model, this is reflected by a single edge in the graph. (That is, in each row of AL , any node is stored not more than once.) It is only relevant that any two nodes of the graph are connected, but it is not important whether the two nodes are connected by one or more than one edge. Additionally, the interface types are not evaluated by method DMP, so they are not reflected in the graph.

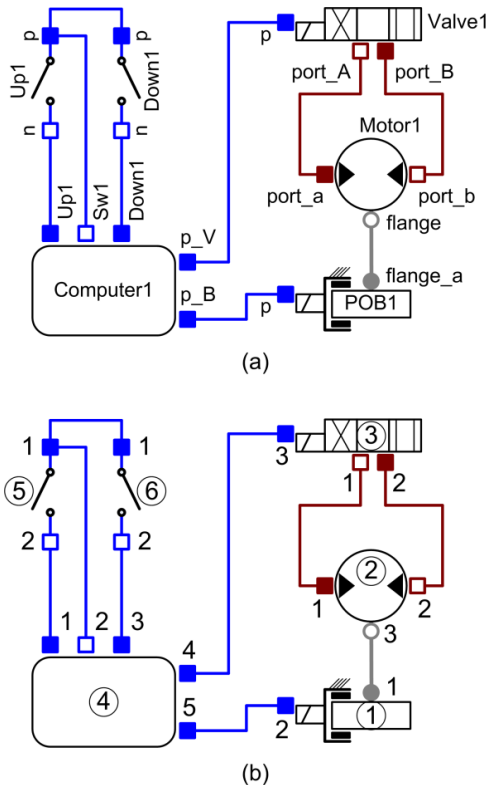


Figure 3. Components and connections in a multi-domain object-oriented model

For illustration, the adjacency list is indicated for a part of the system model depicted in Figure 1. Figure 3 (a) shows the component (node) and interface names, and the indices in (b). The numbering of nodes – encircled in (b) – corresponds to Figure 1. The algorithm that actually prepares an adjacency list is described in subsection 3.1.3 of (Schallert, 2015).

The connections via mechanical flanges, hydraulic ports, electric pins etc. are declared in the model by the `connect()` statements below. They are expressed in

terms of the component and interface names (left column), and in terms of component and interface indices (right column).

1. `connect(POB1.flange_a, Motor1.flange);` (1.1, 2.3)
2. `connect(Motor1.port_a, Valve1.port_A);` (2.1, 3.1)
3. `connect(Motor1.port_b, Valve1.port_B);` (2.2, 3.2)
4. `connect(POB1.p, Computer1.p_B);` (1.2, 4.5)
5. `connect(Valve1.p, Computer1.p_V);` (3.3, 4.4)
6. `connect(Computer1.Sw1, Up1.p);` (4.2, 5.1)
7. `connect(Up1.p, Down1.p);` (5.1, 6.1)
8. `connect(Computer1.Up1, Up1.n);` (4.1, 5.2)
9. `connect(Computer1.Down1, Down1.n);` (4.3, 6.2)

A special case occurs if more than one node is directly or indirectly connected to one and the same interface of a node, as happens for the 6th and 7th connections of the example: `Computer1.Sw1` (4.2) is connected to `Up1.p` (5.1), and in turn `Up1.p` (5.1) is connected to `Down1.p` (6.1). Actually, there is a direct connection between (4.2) and (6.1). It only appears to be indirect, across (5.1), because each `connect()` statement links exactly two nodes. To reflect that a direct connection exists between (4.2) and (6.1), an auxiliary node (14) is introduced. Auxiliary nodes do not represent any real or model object; rather, they are introduced to ensure that coherent sets of nodes are correctly detected by method DMP. An auxiliary node is stored as an additional row in the adjacency list AL .

Table 1 specifies the adjacency list by the node indices. Figure 4 shows the corresponding graph.

Table 1. Adjacency list AL for Figure 3 (b)

1	2	4			
2	1	3			
3	2	4			
4	1	3	5	6	14
5	4	14			
6	4	14			
14	4	5	6		

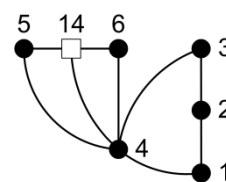


Figure 4. Graph for Figure 3 (b)

3.2 Detection of Minimal Paths

Method DMP is capable of detecting the minimal path set if conditions 1, 2 and 3 defined in subsection 3.1.2 are fulfilled. The detection starts with all system components (nodes) intact. Nodes are then successively removed from the system graph, which corresponds to component failures. The model is simulated to identify if the system still operates or fails.

Articulations can occur in the graph that, if removed, cause disconnection of the graph into several subgraphs. Since only a coherent set of intact nodes can be a minimal path, splitting up the graph at articulations reduces the state space and thus the number of simulations. The lower the density of a system graph is, the more articulations occur within it and thus fewer simulations are required. For completeness, method DMP allows that articulations can also belong to a minimal path.

3.2.1 Detection Algorithm

Figure 5 shows a flow chart of the detection algorithm. It consists of a preparation phase (steps DMP.1 - 4) and the actual, iterative detection process (steps DMP.5 - 17). Steps DMP.3, 8, 11 - 16 refer to lower level algorithms that are described in detail, including code, in (Schallert, 2015). The meaning of the symbols used is as follows:

n_r	number of all components that can fail of a system
n_{loop}	iteration counter of detection process
r_n	node(s) to be removed from a path of array PS_{prev}
PS, PS_{prev}	arrays of path sets in the actual and previous iteration, respectively, of the detection process
$isMinPS, isMinPS_{prev}$	Boolean arrays that store if a path in array PS or PS_{prev} is minimal
np, np_{prev}	number of paths stored in PS and PS_{prev}
SF	array for storing combinations that cause system failure
nsf	number of combinations stored in array SF

In the preparation phase, the necessary data are retrieved from the system model (step DMP.1). Then, the model is simulated to check if the system operates for the set of initially intact components (nodes). To this end, the model output $sysOp$ is evaluated. A monotonous system will operate, and the procedure is continued only in this case (step DMP.2). If the system fails, no minimal path can be detected, and the process is aborted. Next, a graph (adjacency list) of the system model is established (step DMP.3, see 3.1.3). Then, several arrays are initialised (step DMP.4) for the detection process.

At the start of an iteration, the paths detected so far, their number, as well as the information whether they are minimal are assigned to PS_{prev} , np_{prev} and $isMinPS_{prev}$. Arrays PS , $isMinPS$ and the counter np are reset (step DMP.5). Then, n_{loop} is increased by one. Next, combinations are generated from the paths in PS_{prev} . If the i^{th} path, denoted by $PS_{prev}[i, :]$, is minimal (checked in step DMP.7), then it is not further reduced, because any subset of a minimal path causes system failure. If the i^{th} path is not minimal, then all subsets are generated that remove one intact node r_n from the path (step DMP.8): $PS_{prev}[i, :] \setminus \{r_n\}$ for all $r_n \in PS_{prev}[i, :]$ and $r_n \in \{1, nr\}$. If node nr is an articulation of path $PS_{prev}[i, :]$, then the corresponding subgraphs of $PS_{prev}[i, :]$ are generated. Articulations and subgraphs are determined by an algorithm based on depth-first search described by (Tarjan, 1972). Along with each subgraph, the non-articulations of $PS_{prev}[i, :]$ that also belong to the respective subgraph are stored. This information is used later, in step DMP.13, to generate combinations that remove two or more non-articulations from a path, dependent on the simulation result (step DMP.11). Due to monotony of the system, any subset of a path is generated only if it is not a subset of any combination stored in SF that causes system failure. Thus, if no subset is generated from path $PS_{prev}[i, :]$ in step DMP.8, then the system fails for every subset of this path; it is minimal and is marked by $isMinPS_{prev}[i] := true$. The generation of subsets of paths ends after every path in PS_{prev} has been processed, i.e. $i > np_{prev}$ (step DMP.10).

Next (step DMP.11), the model is simulated for every generated combination in order to determine if the system is operating. From the simulation result ($sysOp$), it is first determined which paths in PS_{prev} are minimal. If a path is minimal, it is stored in PS and marked as minimal in $isMinPS$. Then, dependent on whether they cause system operation or failure, the combinations are stored either in PS or in SF , and the respective counter (np or nsf) is increased (step DMP.12).

For those paths in PS that were established due to an articulation and that are no superset of any other path, combinations are generated that remove two or more non-articulations from the original path in PS_{prev} (step DMP.13). This is necessary since articulations can also belong to a minimal path. The system model is then simulated for the generated combinations. Dependent on the simulation result, a combination is stored either in PS or SF (steps DMP.14 and 15).

Next, array PS is tidied up by deleting those paths that are a superset of any other path (step DMP.16). A path can be minimal only if it is not a superset of any other path. If every of the np paths in PS is marked as minimal (step DMP.17), the detection is complete and

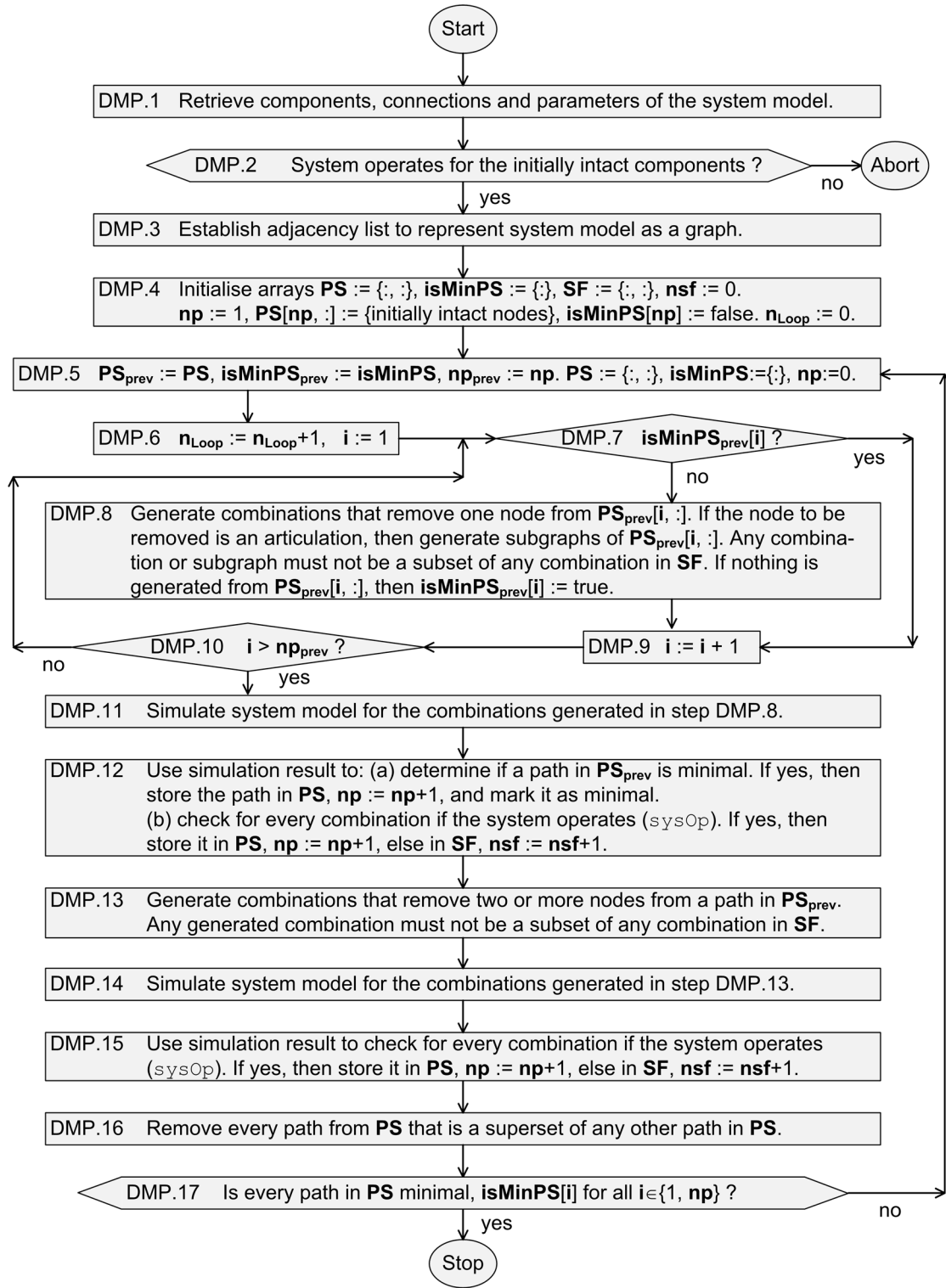


Figure 5. Flow chart of minimal path set detection algorithm DMP

the process ends. Otherwise, the process continues with a new iteration at step DMP.5.

3.2.2 A Minimal Path Set Detection Example

The detection algorithm DMP is illustrated by means of the example graph shown in Figure 6. It is assumed that this graph is deduced from the object-oriented model of any technical system. The minimal path set is assumed as $PS = \{\{1, 2, 3\}, \{4, 5, 6\}, \{1, 3, 4, 6, 7\}\}$.

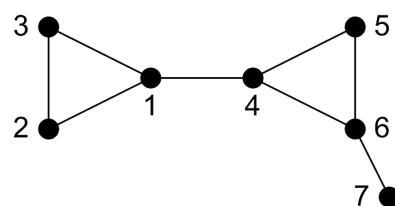


Figure 6. An example graph

The meaning of the symbols used is as follows, as yet not defined:

isArt	indicates if rn is an articulation of the respective path
comb	generated combination of intact nodes
simC	array of generated combinations, input for simulations of the system model
sysOp	array of simulation result (system operates or not) for every combination in $simC$
nonArt	non-articulation nodes of a path in PS_{prev} that also belong to a generated subgraph

The detection proceeds as follows. In the tables, column “row” indicates the progress of the algorithm in terms of “ n_{Loop} ” - “number of $comb$ ”, e.g. 0-1 denotes combination 1 of iteration 0 ($n_{Loop} = 0$).

In the preparation phase it is checked if the system operates when all its components are intact (step DMP.2 in Figure 5). Thus, the simulation input $simC$ is as indicated below in Table 2. At this initial stage, no path has yet been detected and PS_{prev} is empty.

Table 2. Combinations tested (by simulation of system model) at initial stage of detection process

row	PS_{prev}	rn	isArt	comb	comb stored in $simC$		sysOp	nonArt
					no	yes		
0-1	-	-	-	{1, 2, 3, 4, 5, 6, 7}	-	x	x	-

The system operates, so the set of initially intact nodes is stored as a single, non-minimal path ($n_p = 1$) in PS (step DMP.4). SF is empty ($n_{sf} = 0$), Table 3.

Table 3. Path set after initial stage of detection process

PS	isMinPS	SF
{1, 2, 3, 4, 5, 6, 7}	-	-

The process continues with iteration one ($n_{Loop} = 1$). The path data are assigned to PS_{prev} , $isMinPS_{prev}$ and np_{prev} . PS , $isMinPS$ and np are reset (step DMP.5). Combinations are then generated from path $PS_{prev}[1, :]$ as follows (step DMP.8): Node 1 is an articulation. The path splits into two subgraphs {2, 3} and {4, 5, 6, 7} due to the removal of node 1. The non-articulations of the original path $PS_{prev}[1, :]$ that also belong to the respective subgraphs are {2, 3} and {5, 7}. Node 2 is not an articulation, thus a combination is generated by removing node 2 from $PS_{prev}[1, :]$, and likewise for nodes 3, 5 and 7. Altogether, ten combinations are generated for simulation of the system model, Table 4.

The simulation result of step DMP.11 (column $sysOp$) indicates that path $PS_{prev}[1, :]$ is not minimal, because the system operates for subsets of it, namely for those in rows 1-2, 1-3, 1-4, 1-5, 1-7, 1-8 and 1-10. These combinations are stored as paths in PS , $n_p = 7$. The other combinations in rows 1-1 and 1-6 are stored in SF , $n_{sf} = 2$ (step DMP.12). The one in row 1-9, {7}, is not stored in SF as it is a subset of {5, 6, 7}.

At this stage, two of the seven paths in PS are not supersets of any other path, namely rows 1-2 and 1-5 in Table 4 (marked bold). Other paths can exist that include some of the articulations of the original path $PS_{prev}[1, :]$. To assure that such paths are detected, further combinations that are no superset of any path in PS - in this case {4, 5, 6, 7} and {1, 2, 3} - must be generated (step DMP.13). Such combinations remove as many non-articulations from the original path as non-superset paths were deduced from it, namely two (rows 1-2 and 1-5) in the case of $PS_{prev}[1, :]$. To avoid generating supersets, one node of every set of non-articulations, {5, 7} and {2, 3}, is removed from the original path, respectively. Thus, the combinations $PS_{prev}[1, :] \setminus \{2, 5\}$, $PS_{prev}[1, :] \setminus \{2, 7\}$, $PS_{prev}[1, :] \setminus \{3, 5\}$ and $PS_{prev}[1, :] \setminus \{3, 7\}$ are generated, as listed in rows 1-11 through 1-14, Table 5.

Due to the simulation result, three more paths are stored in PS , $n_p = 7 + 3 = 10$, and one more combination in SF , $n_{sf} = 2 + 1 = 3$ (steps DMP.14 and 15). The total number of simulations so far is $n_{sim} =$

$1 + 10 + 4 = 15$. Supersets of paths are removed from PS , which leads to $n_p = 5$ paths remaining (in Table 6) after completion of step DMP.16.

Table 6. Path set PS and combinations that cause system failure SF , as existent after 1st iteration of process

PS	isMinPS	SF
{1, 2, 3}	-	{1, 2, 4, 6, 7}
{1, 2, 4, 5, 6}	-	{2, 3}
{1, 3, 4, 5, 6}	-	{5, 6, 7}
{1, 3, 4, 6, 7}	-	
{4, 5, 6, 7}	-	

Since none of the paths in PS is marked as minimal (step DMP.17), the process continues with a second iteration ($n_{Loop} = 2$). The path data are assigned to PS_{prev} , $isMinPS_{prev}$ and np_{prev} . PS , $isMinPS$ and np are reset (step DMP.5). Then, combinations are generated (step DMP.8) from each of the $np_{prev} = 5$ paths in PS_{prev} as listed in Table 7. Three combinations are generated from $PS_{prev}[1, :] = \{1, 2, 3\}$, but only one is stored in $simC$ for simulation. The other two are not stored in $simC$ because they are a subset of a combination in SF , as indicated in rows 2-1 and 2-3. If a combination causes system failure, every subset of it causes system failure as well due to system monotony.

Any combination is stored only once in $simC$, as indicated in row 2-12, for instance. Eight combinations are stored altogether for simulation in step DMP.11.

The simulation result (column `sysOp` in Table 7) indicates that $\text{PS}_{\text{prev}}[1, :] = \{1, 2, 3\}$ and $\text{PS}_{\text{prev}}[4, :] = \{1, 3, 4, 6, 7\}$ are minimal, because the system fails for every respective subset. These paths are stored in `PS` and marked as minimal in `isMinPS` (step DMP.12). In addition, two non-minimal paths, $\{4, 5, 6\}$ in row 2-5 and $\{1, 4, 5, 6\}$ in row 2-6, are stored in `PS`; the latter will be removed in step DMP.16.

It is not necessary in this iteration to generate combinations that remove two or more non-articulations from any path in PS_{prev} . The reason is: At most one subgraph that causes system operation is deduced from any path in PS_{prev} . In the case of $\text{PS}_{\text{prev}}[2, :] = \{1, 2, 4, 5, 6\}$, subgraphs $\{2\}$, $\{4, 5, 6\}$ and $\{1, 2\}$, $\{5, 6\}$ are generated due to articulations 1 and 4, respectively. The system operates only for $\{4, 5, 6\}$. In order to generate every combination from $\text{PS}_{\text{prev}}[2, :]$ that is no superset of $\{4, 5, 6\}$, it is sufficient to remove one non-articulation from $\text{PS}_{\text{prev}}[2, :]$. These combinations are generated already in step DMP.8, as Table 7 shows (rows 2-9 and 2-10).

Thus, $n_p = 4$ paths are stored in `PS` of which two are minimal. $n_{sf} = 3 + 3 = 6$ combinations are stored in `SF`. The total number of simulations so far is $n_{\text{sim}} = 15 + 8 = 23$. $n_p = 3$ paths remain in `PS` (see Table 8) after removal of supersets in step DMP.16.

Table 8. Path set `PS` and combinations that cause system failure `SF`, as existent after 2nd iteration of process

PS	isMinPS	SF
$\{1, 2, 3\}$	x	$\{1, 2, 4, 5\}$
$\{1, 3, 4, 6, 7\}$	x	$\{1, 2, 4, 6, 7\}$
$\{4, 5, 6\}$	-	$\{1, 3, 4, 5\}$
		$\{1, 3, 4, 6\}$
		$\{2, 3\}$
		$\{5, 6, 7\}$

Since not all paths are marked as minimal, the process enters a third iteration, $n_{\text{loop}} = 3$. Again, the path data are assigned to PS_{prev} , `isMinPSprev` and `npprev`. `PS`, `isMinPS` and `np` are reset. Then, combinations are generated only for the non-minimal path $\text{PS}_{\text{prev}}[3, :] = \{4, 5, 6\}$ in step DMP.8. As Table 9 shows, every generated combination is a subset of any combination in `SF`. This means that $\text{PS}_{\text{prev}}[3, :]$ is also minimal, and no further simulations are necessary. Thus, the process is complete. The minimal path set of the example system (Figure 6) is correctly detected:

Table 10. Minimal path set detected after 3rd iteration

PS	isMinPS
$\{1, 2, 3\}$	x
$\{1, 3, 4, 6, 7\}$	x
$\{4, 5, 6\}$	x

Next, the probability of system operation (or failure) is computed. For illustration, equation 3 is evaluated for the detected minimal path set. With the component reliabilities $R_i = R_i(t)$, the probabilities of the minimal paths are:

$$P(\text{PS}_1) = R_1 R_2 R_3, \quad P(\text{PS}_2) = R_1 R_3 R_4 R_6 R_7, \\ P(\text{PS}_3) = R_4 R_5 R_6.$$

For the 2nd order intersections, the probabilities are:

$$P(\text{PS}_1 \wedge \text{PS}_2) = R_1 R_2 R_3 R_4 R_6 R_7, \\ P(\text{PS}_1 \wedge \text{PS}_3) = R_1 R_2 R_3 R_4 R_5 R_6, \\ P(\text{PS}_2 \wedge \text{PS}_3) = R_1 R_3 R_4 R_5 R_6 R_7.$$

The probability of the single 3rd order intersection is:

$$P(\text{PS}_1 \wedge \text{PS}_2 \wedge \text{PS}_3) = R_1 R_2 R_3 R_4 R_5 R_6 R_7.$$

Employing these products, equation 3 reads

$$R_{\text{sys}}(t) = R_1 R_2 R_3 + R_1 R_3 R_4 R_6 R_7 + R_4 R_5 R_6 \\ - (R_1 R_2 R_3 R_4 R_6 R_7 + R_1 R_2 R_3 R_4 R_5 R_6 \\ + R_1 R_3 R_4 R_5 R_6 R_7) + R_1 R_2 R_3 R_4 R_5 R_6 R_7.$$

If it is assumed that $\lambda_i = 10^{-2}/\text{h}$ and $t = 1\text{h}$, thus $R = R_i = 0.990$, then the probability of system operation is $R_{\text{sys}}(1\text{h}) = 2R^3 + R^5 - 3R^6 + R^7 = 0.99922$ or likewise, the probability of system failure is $F_{\text{sys}}(1\text{h}) = 1 - R_{\text{sys}}(1\text{h}) = 7.8 \cdot 10^{-4}$.

3.2.3 Proof and boundary effort of detection method

The minimal path set detection method DMP gives a complete result when applied to any multi-domain object-oriented system model that fulfills the conditions 1, 2 and 3 stated in subsection 3.1.2. In addition, the method is finite which means that it terminates when applied to any such model. The completeness and finiteness are proven in the following. The upper and lower bounds of the required computing effort are also derived.

Completeness. Consider a detection method that merely exploits the monotony of the analysed system. It starts with a set of all nodes as the initial path. Every combination is generated that removes a single node from the path. It is tested for each (by simulation of the model) if the system operates. Those combinations that cause system operation constitute a set of paths. In the set of paths, only those are kept that are no superset of any other, because only a non-superset path can be minimal. Thus, a complete path set of the system exists at the end of an iteration of the detection method.

A next iteration is entered. All combinations are generated that remove a single node from every path in the set of the previous iteration. In so doing, subsets of combinations that cause system failure are omitted. Due to monotony, if a combination causes system failure, the system remains failed if any node is removed from that combination. Again, testing the generated combinations leads to a complete set of paths, a next iteration is entered with all non-superset paths of the set, and so on. At some point it is found that the system fails on removal of any node from a path. Such a path is minimal by definition. The method

continues reducing the other paths until all in the set of paths are minimal. Since every path was reduced by a single node from one iteration to the next, it is obvious that the method gives the complete minimal path set.

In addition to exploiting system monotony, method DMP benefits from the fact that only a coherent set of intact nodes can be a minimal path. Evaluation of the system graph hence reduces the number of simulations required for minimal path set detection.

A path is split into subgraphs at the articulations of the path. In this way, subgraphs are generated for every path that exists in the path set of the previous iteration of DMP. If two or more subgraphs of a path cause system operation, all combinations are generated that remove as many non-articulations from that path, as subgraphs were deduced from it that cause system operation. (Two or more nodes are removed, because all combinations that reduce a path by one of its non-articulations have been generated and tested in a preceding step.) In so doing, for the generated combinations to be no superset of any of the subgraphs of that path, each non-articulation removed from the path belongs to exactly one of its subgraphs. Thus, if any such combination causes system operation, it exists in the path set at the end of an iteration of DMP. It follows that the path set at the end of any iteration is complete, and hence method DMP is complete.

Finiteness. DMP commences with a set of all nodes of a monotonous system as the initial path. It tests if the system still operates for subsets of a (the initial or other) path. Every subset removes one or more nodes from a path. Nodes are never added to a path from one iteration to the next. DMP repeats gradually removing nodes until the system fails for all subsets of a path, i.e. if that path is minimal. For those paths not yet identified as minimal, subsets of them are generated until they be reduced no further without causing system failure, i.e. until every path is minimal. Then, the process ends. If every single node of a system constitutes a minimal path, the process ends after all nodes are failed. Due to monotony, a system fails if all its nodes fail. Thus, method DMP clearly terminates.

simulations in each iteration n_{Loop} of DMP is the binomial $\mathbf{C}(nr, n_{Loop})$. DMP iterates until all nodes are failed, i.e. $n_{Loop} = nr$. The total number of simulations is thus $\sum_{n_{Loop}=0}^{nr} \mathbf{C}(nr, n_{Loop}) = 2^{nr}$, the same as a “brute force” approach needs.

The lowest effort occurs if a system operates only with all its nodes intact (single minimal path that comprises all nodes of the system). Irrespective of the density of the system graph, DMP runs until iteration $n_{Loop} = 1$ is completed. The total number of simulations is hence $\sum_{n_{Loop}=0}^1 \mathbf{C}(nr, n_{Loop}) = 1 + nr$.

The number of simulations required by DMP is thus bound by the upper limit 2^{nr} and lower limit $1 + nr$. Between these bounds, the actual effort depends on the density of the system graph, the number of minimal paths and number of nodes thereof. The more effort is saved, the lower the density of a system graph is.

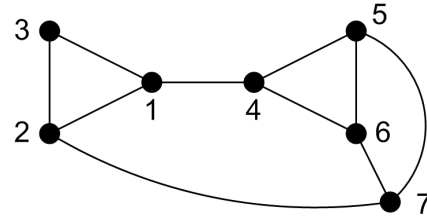


Figure 7. An example graph, higher density than Figure 6. For illustration, Table 11 lists the number of simulations n_{sim} for three detection cases. The number of edges and density (equation 1) of the respective graphs are denoted by E and d . The number of nodes is $N = nr = 7$ for all cases. n_{sim} is stated for method DMP, as well as a method that exploits the system “monotony only” (no evaluation of the graph), as described. Case 2 corresponds to the example in subsection 3.2.2. Case 1 relates to the same graph, but the system has one less minimal path. Case 3 assumes the same minimal path set as case 2, but the graph has a higher density (Figure 7). The comparison shows that the fewer minimal paths exist and the lower the density of the graph, the smaller effort required by DMP. With an increasing number of minimal paths and graph density, the effort of DMP

Table 11. Comparison of effort of minimal path set detection for three cases

case	system graph	E	d	PS	n_{sim}	
					monotony only	DMP
1	Figure 6	8	0.381	$\{1, 2, 3\}, \{4, 5, 6, 7\}$	35	15
2	Figure 6	8	0.381	$\{1, 2, 3\}, \{4, 5, 6\}, \{1, 3, 4, 6, 7\}$	40	23
3	Figure 7	10	0.476	$\{1, 2, 3\}, \{4, 5, 6\}, \{1, 3, 4, 6, 7\}$	40	34

Effort. For illustration of the highest computing effort, consider a complete system graph that includes no articulations. Removal of any node gives a subsequent smaller complete graph. In addition, consider that every single node of the nr nodes of the system graph constitutes a minimal path. Then, the number of

approaches that of the “monotony only” method. A “brute force” method neither evaluates the monotony of a system nor its graph; it thus requires $2^{nr} = 128$ simulations for each case.

4 Conclusions

This paper contributes a method called DMP for detection of the minimal path set of any fault-tolerant system that is represented as a multi-domain object-oriented model. DMP can be employed throughout the system development process to keep the safety analysis up-to-date with design iterations. This is meaningful particularly if multi-domain object-oriented modelling is used already in systems engineering. DMP enhances the scope of application of a model while permitting all other simulation studies that originally motivated implementation of the model to be conducted.

DMP belongs to the class of state-space simulations. Evaluation of the system graph reduces the number of simulations required, thus ensuring feasibility of DMP. It has been successfully tested on large, realistic models of safety relevant aircraft systems, as described in (Schallert, 2015).

It must be beared in mind that all model-based safety analysis methods capture only those phenomena that are covered in the modelling. A model is always an abstraction of a real system and might hence be incomplete. Then again, the DMP method ensures that all relevant failure conditions, at least in so far as modelled, are captured.

Acknowledgements

This research has received funding from the European Union's 7th Framework Programme (FP7/2007-2013) for the CleanSky Joint Technology Initiative under grant agreement CSJU-GAN-SGO-2008-001.

References

- A. Birolini. *Reliability Engineering – Theory and Practice* (Fifth Edition). Springer-Verlag Berlin Heidelberg, 2007.
- P. Bunus, K. Lunde. Supporting Model-Based Diagnostics with Equation-Based Object-Oriented Languages. *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)*, pp. 121-130, Paphos, Cyprus, 2008.
- R. Diestel. *Graph Theory (Graduate Texts in Mathematics)*, Springer-Verlag, 2010.
- H. Elmqvist, S. E. Mattsson, M. Otter. Modelica extensions for Multi-Mode DAE-Systems. *Proceedings of the 10th International Modelica Conference*, pp. 183-193, Lund, Sweden, 2014. doi: 10.3384/ECP14096183
- A. Meyna, B. Pauli. *Taschenbuch der Zuverlässigkeits- und Sicherheitstechnik*. Carl Hanser Verlag München Wien, 2003. In German.
- C. Schallert. Incorporation of Reliability Analysis Methods with Modelica. *Proceedings of the 6th International Modelica Conference*, pp. 103-112, Bielefeld, Germany, 2008.
- C. Schallert. Inclusion of Reliability and Safety Analysis Methods in Modelica. *Proceedings of the 8th International Modelica Conference*, pp. 616-627, Dresden, Germany, 2011. doi: 10.3384/ECP11063616
- C. Schallert. A Safety Analysis via Minimal Path Sets Detection for Object-Oriented Models. *Safety and Reliability: Methodology and Applications (editors: Nowakowski et al.)*, CRC Press/Balkema, ISBN: 978-1-315-73697-6, 2014.
- C. Schallert. *Integrated Safety and Reliability Analysis Methods for Aircraft System Development using Multi-Domain Object-Oriented Models*, 2015 (to appear).
- R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2), pp. 146-160, 1972.
- F. van der Linden. General fault triggering architecture to trigger model faults in Modelica using a standardized blockset. *Proceedings of the 10th International Modelica Conference*, pp. 427-436, Lund, Sweden, 2014. doi: 10.3384/ECP14096427
- Y. Papadopoulos, J. McDermid, R. Sasse, G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering and System Safety*, Vol. 71, pp. 229 - 247, 2001.
- D. Zimmer, M. Otter, H. Elmqvist, G. Kurzbach. Custom Annotations: Handling Meta-Information in Modelica. *Proceedings of the 10th International Modelica Conference*, pp. 173-182, Lund, Sweden, 2014. doi: 10.3384/ECP14096173

Table 4. Combinations tested during 1st iteration of detection process

row	PS _{prev}	rn	isArt	comb	comb stored in simC		sysOp	nonArt
					no	yes		
1-1	{1, 2, 3, 4, 5, 6, 7}	{1}	x	{2, 3}	-	x	-	{2, 3}
1-2		{1}	x	{4, 5, 6, 7}	-	x	x	{5, 7}
1-3		{2}	-	{1, 3, 4, 5, 6, 7}	-	x	x	-
1-4		{3}	-	{1, 2, 4, 5, 6, 7}	-	x	x	-
1-5		{4}	x	{1, 2, 3}	-	x	x	{2, 3}
1-6		{4}	x	{5, 6, 7}	-	x	-	{5, 7}
1-7		{5}	-	{1, 2, 3, 4, 6, 7}	-	x	x	-
1-8		{6}	x	{1, 2, 3, 4, 5}	-	x	x	{2, 3, 5}
1-9		{6}	x	{7}	-	x	-	{7}
1-10		{7}	-	{1, 2, 3, 4, 5, 6}	-	x	x	-

Table 5. Combinations tested during 1st iteration that remove two non-articulations from $PS_{prev}[1, :]$

row	PS_{prev}	rn	comb	comb stored in simC		sysOp
				no	yes	
1-11	{1, 2, 3, 4, 5, 6, 7}	{2, 5}	{1, 3, 4, 6, 7}	-	x	x
1-12		{2, 7}	{1, 3, 4, 5, 6}	-	x	x
1-13		{3, 5}	{1, 2, 4, 6, 7}	-	x	-
1-14		{3, 7}	{1, 2, 4, 5, 6}	-	x	x

Table 7. Combinations tested during 2nd iteration of detection process

row	PS_{prev}	rn	isArt	comb	comb stored in simC		sysOp	nonArt
					no	yes		
2-1	{1, 2, 3}	{1}	-	{2, 3}	$\subseteq\{2, 3\}$	-	-	-
2-2		{2}	-	{1, 3}	-	x	-	-
2-3		{3}	-	{1, 2}	$\subseteq\{1, 2, 4, 6, 7\}$	-	-	-
2-4	{1, 2, 4, 5, 6}	{1}	x	{2}	$\subseteq\{1, 2, 4, 6, 7\}$	-	-	-
2-5		{1}	x	{4, 5, 6}	-	x	x	{5, 6}
2-6		{2}	-	{1, 4, 5, 6}	-	x	x	-
2-7		{4}	x	{1, 2}	$\subseteq\{1, 2, 4, 6, 7\}$	-	-	-
2-8		{4}	x	{5, 6}	$\subseteq\{5, 6, 7\}$	-	-	-
2-9		{5}	-	{1, 2, 4, 6}	$\subseteq\{1, 2, 4, 6, 7\}$	-	-	-
2-10		{6}	-	{1, 2, 4, 5}	-	x	-	-
2-11		{1, 3, 4, 5, 6}	{1}	x	{3}	$\subseteq\{2, 3\}$	-	-
2-12	{1}		x	{4, 5, 6}	exists in simC	-	x	-
2-13	{3}		-	{1, 4, 5, 6}	exists in simC	-	x	-
2-14	{4}		x	{1, 3}	exists in simC	-	-	-
2-15	{4}		x	{5, 6}	$\subseteq\{5, 6, 7\}$	-	-	-
2-16	{5}		-	{1, 3, 4, 6}	-	x	-	-
2-17	{6}		-	{1, 3, 4, 5}	-	x	-	-
2-18	{1, 3, 4, 6, 7}	{1}	x	{3}	$\subseteq\{2, 3\}$	-	-	-
2-19		{1}	x	{4, 6, 7}	$\subseteq\{1, 2, 4, 6, 7\}$	-	-	-
2-20		{3}	-	{1, 4, 6, 7}	$\subseteq\{1, 2, 4, 6, 7\}$	-	-	-
2-21		{4}	x	{1, 3}	exists in simC	-	-	-
2-22		{4}	x	{6, 7}	$\subseteq\{5, 6, 7\}$	-	-	-
2-23		{6}	x	{1, 3, 4}	-	x	-	{3}
2-24		{6}	x	{7}	$\subseteq\{5, 6, 7\}$	-	-	-
2-25		{7}	-	{1, 3, 4, 6}	exists in simC	-	-	-
2-26	{4, 5, 6, 7}	{4}	-	{5, 6, 7}	$\subseteq\{5, 6, 7\}$	-	-	-
2-27		{5}	-	{4, 6, 7}	$\subseteq\{1, 2, 4, 6, 7\}$	-	-	-
2-28		{6}	x	{4, 5}	-	x	-	{4, 5}
2-29		{6}	x	{7}	$\subseteq\{5, 6, 7\}$	-	-	-
2-30		{7}	-	{4, 5, 6}	exists in simC	-	x	-

Table 9. 3rd iteration of detection process (no combinations tested)

row	PS_{prev}	rn	isArt	comb	comb stored in simC		sysOp	nonArt
					no	yes		
3-1	{4, 5, 6}	{4}	-	{5, 6}	$\subseteq\{5, 6, 7\}$	-	-	-
3-2		{5}	-	{4, 6}	$\subseteq\{1, 3, 4, 6\}$	-	-	-
3-3		{6}	-	{4, 5}	$\subseteq\{1, 2, 4, 5\}$	-	-	-

High-fidelity Modelling of Self-regulating Pneumatic Valves

Alexander Pollok¹ Francesco Casella²

¹Institute of System Dynamics and Control, German Aerospace Center (DLR), Germany,
alexander.pollok@dlr.de

²Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
francesco.casella@polimi.it

Abstract

In conventional aircraft energy systems, self-regulating pneumatic valves (SRPVs) are used to control the pressure and mass flow of the bleed air. The dynamic behavior of these valves is complex and dependent on several physical phenomena. In some cases, limit cycles can occur, deteriorating performance. This paper presents a complex multiphysical model of SRPVs implemented in Modelica. First, the working-principle is explained, and common challenges in control-system design-problems related to these valves are illustrated. Then, a Modelica-model is presented in detail, taking into account several physical domains. It is shown, how limit cycle oscillations occurring in aircraft energy systems can be represented with this model. Finally, some multi-domain interactive effects are described.

Keywords: Modelica, Thermofluid, Modeling, Friction, Electrohydraulic, Hydraulic

1 Introduction

In applications related to process control often relatively simple valve models are used. They are based on flow coefficients, and relate mass flow to pressure drop by the use of a quadratic relationship. This helps keeping the system model at a low-order, benefitting understanding as well as control design. Most of the time, these simple models are accurate enough, and all relevant dynamics are included.

There are however applications, where simple models are inadequate. This can be the case, if high accuracy is needed, when choking occurs, or when internal valve phenomena are relevant. Neglecting of these cases, and the utilization of an inadequate model can lead to unwanted behavior in the controlled system: Valve dynamics often contain nonlinearities like stiction, backlash and deadband, which in turn can lead to oscillations (Choudhury et al., 2006).

Indeed, according to Bialowski (1993), about 30% of controlled loops in the process industry are oscillating. In Desborough and Miller (2002), 26.000 PID con-

trollers in the process industry are surveyed: 16% are classified as excellent, 16% as acceptable, 22% as fair, 10% as poor, and 36% run in open-loop.

In aircraft, SRPVs are used to control the pressure and flow rate of the engine bleed air. An illustration of the working principle can be found in Figure 1, more detailed descriptions can be found in Section 2.

SRPVs operate under harsh conditions inside the engine nacelle. Since several SRPVs are operated in-line, their dynamic behavior has to be tuned so as to avoid the occurrence of limit cycles. This can be done in situ, but the associated costs are substantial. Being able to predict the system behavior better during the design phase would reduce those costs considerably, but for a sufficient level of prediction-accuracy a high-fidelity model is needed.

Related research has been done by several authors. Beater (2000) presented a simple model of an electrohydraulic valve in Modelica and HyLib. In Beater and Clauß (2003), a pneumatic drive system is modelled in Modelica, combining pneumatic, mechanical and electronic domains. A free-piston-engine modelled in Modelica is described in Pohl and Gräf (2005), containing detailed submodels of several physical domains. Pujana-Arrese et al. (2007) presented a Modelica-model of a pneumatic muscle, combining fluid modelling with the mechanical system of kinematics.

The goal of this paper is to demonstrate how high-fidelity multi-physical models of self-regulating pneumatic valves can be developed in the object-oriented equation-based modelling-language Modelica. It is structured as follows: In Section 2, the Modelica model for SRPVs is presented and the motivations for modelling choices are explained, subdivided into the different physical domains. Libraries, models and implementations that are used in this work are mentioned. In Section 3, exemplary model outputs are shown, and a number of emerging phenomena are discussed. The paper is concluded in Section 4.


```

"flow cross section";
output Modelica.SIunits.MassFlowRate
  m_flow "mass flow";
Real minp "lower pressure";
Real maxp "upper pressure";
Real prcrit "critical pressure ratio";
Real ratio "actual pressure ratio";
Real psi "flow function";
Real sig "flow direction";
Real d "density at actual upstream";
algorithm
  minp := min(pu, pd);
  maxp := max(pu, pd);
  prcrit := (2/(kappa+1))^(1/(kappa-1))
    * (kappa/(kappa+1))^(1/2);
  ratio := minp/maxp;
  psi := if ratio < prcrit
    then (2/(kappa+1))^(1/(kappa-1))
    * (kappa/(kappa+1))^(1/2)
    else (kappa*ratio^(1/kappa)
    * (ratio^(1/kappa)-ratio)
    / (kappa-1))^(1/2);
  sig := sign(pu-pd);
  d := if sig > 0 then du else dd;
  m_flow := psi*A*sig*(d*2*maxp)^(1/2);
end nozzle_flow;
    
```

Fluids moving through a butterfly valve at high velocities induce a fluiddynamic torque on the valve disk. This generates an interesting coupling between the fluid and mechanic domains of a valve model. For the calculation of the torque, two approaches are often used: one based on the pressure difference, one based on the fluid velocity. In Sollicec and Danbon (1999), the different approaches are compared. We use the classical approach based on pressure difference, as the pressure difference is more clearly defined than the fluid velocity in the context of lumped parameter models. Here, the torque T is calculated as:

$$T(\alpha) = K(\alpha) \cdot \Delta P \cdot D^3 \quad (1)$$

where K is the torque coefficient, ΔP is the pressure difference, α is the valve angle and D is the valve diameter. A spline-based approach is used to describe the dependency between torque coefficient and valve angle. A Modelica multibody connector provides the valve angle and feeds back the induced fluiddynamic torque.

2.3 Actuator model

Two actuator models as described in Section 2.1 are needed, for two different implementations of the second control loop. Accordingly, one partial model together with two extending models was created. The Modelica diagram of the base model can be seen in Figure 4.

Three physical domains are significant for the modelling of the valve actuator: the fluid dynamics inside the chambers, the multi-body mechanics of the mechanism, and the thermal behavior of the parts. They are connected through the piston and chamber components,

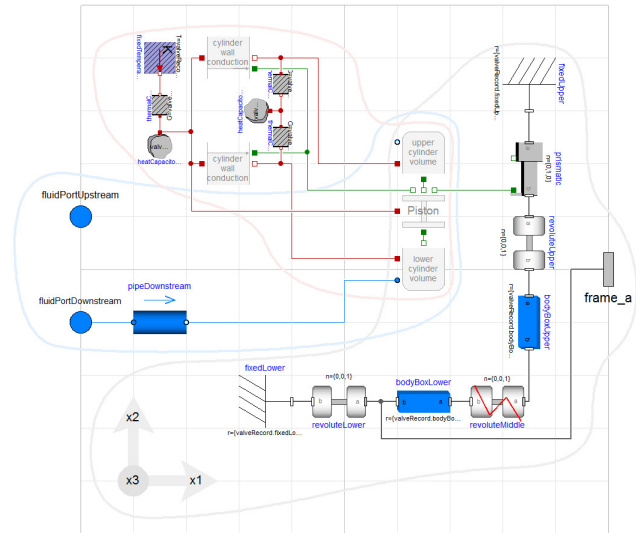


Figure 4. Modelica component layer of the (partial) valve actuator base model

where all domains have considerable influence. The domains are indicated in Figure 4 through colored lines.

2.3.1 Mechanical domain

The core of the mechanical domain is the piston-model, where a one-dimensional force balance over the piston is calculated, see Equation 2. The occurring forces are commented in the following:

$$F_{pressure_{upper}} + F_{pressure_{lower}} + F_{constraint} + F_{friction} + F_{d'alembert} + F_{joint} = 0 \quad (2)$$

Pressure forces:

The piston model and both chamber models are connected by translational mechanical connectors. In this way, the position and the forces generated by fluid pressure are exchanged.

Constraining forces:

Based on the construction, the movement allowance of the piston is limited. To represent this, stiff quadratic spring forces are implemented. These come into effect as soon as the end of the stroke is reached.

Friction force:

The friction forces between piston and cylinder are mainly responsible for unwanted stiction-effects. Detailed modelling of friction phenomena is therefore necessary. Furthermore, a simple model based on two static and dynamic friction coefficients is numerically unfavourable when the piston position is used as a state. In this work, we used the Lund-Grenoble (Lu-Gre) friction model (De Wit et al.,

1995). It is a detailed model of friction with internal states that represent the deflection of the bristles (micro-bumps in the material surface). The implementation in Modelica was done according to Aberger and Otter (2002), but instead of rotatory coordinates, translative coordinates were used. An example trajectory of friction force over piston velocity can be seen in Figure 5.

d'Alembert force:

The d'Alembert force, or inertial force, of the piston is calculated by deriving the position w.r.t. time two times and multiplying with its mass. Of course, this makes the system quite stiff from a numerical point of view, but then, there are solvers of production-quality available to handle stiff systems.

Joint force:

The joint force is the linking force between the translative piston dynamics and the planar dynamics of the mechanism. The prismatic joint model of the multibody library provides the interface.

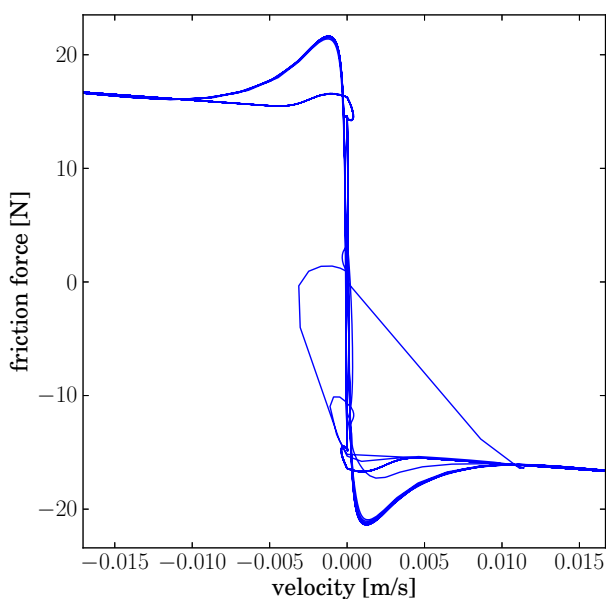


Figure 5. A trajectory (friction force w.r.t. velocity) of the Lund-Grenoble friction model

For the dynamics of the mechanism, the Modelica Multibody library as presented in Otter et al. (2003) is used. With this library, the mechanism can be represented exactly; also an extension to alternative designs can be done with little effort. Unfortunately, nonlinear systems of equations cannot be avoided at this point.

2.3.2 Fluid domain

For the air in the valve actuator, high-speed fluid effects can be neglected. Consequently, the Modelica fluid li-

brary as presented in Casella et al. (2006) is used wherever possible.

Both valve chambers correlate to variable volume models, something not yet implemented in the Modelica fluid library. The governing equations of a variable volume model are a generalisation of the standard volume model equations, and take the form of Equation 3, with the density ρ , the volume V , and $\phi \in (1, u, \mathbf{x})$ representing mass, energy and substance balance respectively.

$$\frac{d}{dt}(\phi \cdot \rho \cdot V) = \sum flow + \sum source \quad (3)$$

In the case of the energy-balance, mechanical work on the cylindrical chamber volume now creates an interesting interaction between the fluid and mechanical domain. The implementation in Modelica can be seen in Listing 2.

Listing 2. Extract of Modelica code for lower variable volume model

```
//translative mechanics interface
medium.p = - flange.f/area;
pos = flange.s;
volume = volume_0 + area*pos;

//mass balance
mass = volume*medium.d;
der(volume*medium.d)
= sum(fluidPort.m_flow);

//energy balance (dU = dQ + dW)
der(volume*medium.d*medium.u)
= sum(fluidPort[i].m_flow * noEvent(
  actualStream(fluidPort[i].h_outflow))
  for i in 1:ninf)
- medium.p*der(volume)
+ heatPort.Q_flow;

//substance balance
der(volume*medium.d*medium.Xi)
= sum(fluidPort[i].m_flow * noEvent(
  actualStream(fluidPort[i].Xi_outflow))
  for i in 1:ninf);
```

2.3.3 Thermal domain

The thermal effects in self-regulating pneumatic valve systems are largely dominated by the advection in the air. This is obviously already included in the fluid modelling. Nonetheless, conduction through the solid components still has to be modelled if high-fidelity results are necessary.

On the thermal side, the model is structured as follows: The environment is modelled as boundary condition of constant temperature. The actuator cylinder wall and piston are both modelled as thermal masses. A further discretization is discarded based on the high internal conductivity of the used materials. The energy dissipated by friction is added to the piston wall. Between

the fluid volumes and the piston mass, as well as between the cylinder wall and the environment, constant thermal conductances are assumed. Between the fluid volumes and the cylinder wall, the thermal conductance is dependent on the wetted area, which is in turn dependent on the piston position.

As a consequence, a heat-conduction component was composed that connects heat conductivity with the piston position. The remainder was modelled using the Modelica thermal heat transfer library, the details of which are described by Tiller (2001).

In Figure 6, the structure of the thermal model is illustrated.

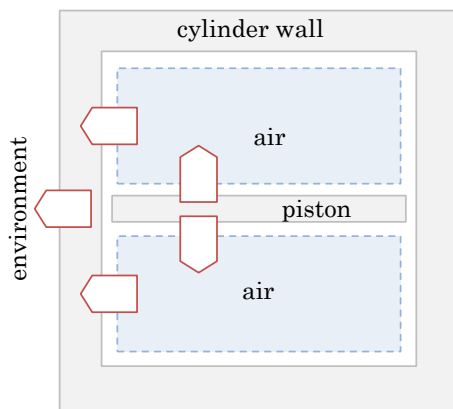


Figure 6. Thermal structure of the valve actuator

2.4 Statistics

The resulting models of valve and actuator feature (0+10) states, (21+161) time-varying variables and (0 + {3}) nonlinear systems of equations respectively.

3 Results and Discussion

3.1 Application

To use the model for simulations, a set of parameters has to be defined. Most of them have a geometrical meaning and can simply be taken from the specifications. For accurate results, there are however three separate measurements to be done:

3.1.1 Friction

In the calculation of the piston-friction as appearing in Equation 2, the Lund-Grenoble (Lu-Gre) friction model (De Wit et al., 1995) is used. In this model, the friction characteristics are defined by 6 constants. These have to be obtained from experiments or looked up in literature, based on the material-pairing.

3.1.2 Aerodynamic Torque

The aerodynamic torque as described in Equation 1 is dependent on the angle of the valve-disc. This dependency differs somewhat based on the geometry, but can often be estimated by CFD-calculations.

3.1.3 Mass Flow Characteristic

Butterfly Valves feature a S-shaped dependency between mass flow and valve angle. Like the aerodynamic torque, this dependency is only somewhat similar between valve-models. Therefore, CFD-calculations or experiments have to be deployed.

3.2 Limit Cycle Oscillations

For reasons of confidentiality, no actual valve setups or associated measurements can be presented here. Instead, a simpler composition is shown, where two valves are used to reduce the pressure in a pipe. The Modelica diagram of the composition can be seen in Figure 7. The pipe models are based on the gas dynamics library as presented by Sielemann (2012b). Each pipe-component represents a pipe of 20 meters length and a diameter of 0.1 m, totalling at a length of 80 meters and a volume of around 630 liters.

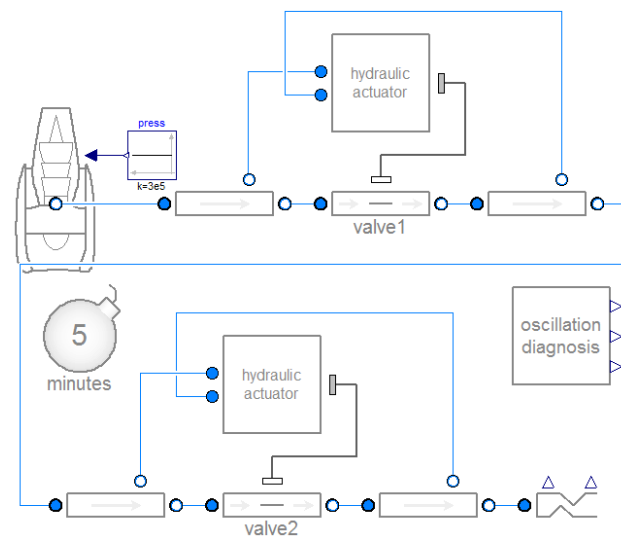


Figure 7. Modelica diagram of oscillation test case

As boundary conditions, the input pressure (left side) is set to 3 bars, while the right boundary is modelled as a quadratic resistance, normalized to a fluid velocity of $10 \frac{m}{s}$ at a pressure of 1 bar. The valve actuators are run in pneumatic-mode and set to regulate the downstream pressure to 2 and 1 bars respectively.

When the composite model is simulated, limit cycle oscillations occur. These are displayed in Figure 8. For both valves, the piston gets stuck at the outmost deflection, until the restoring forces are high enough to overcome the friction forces.

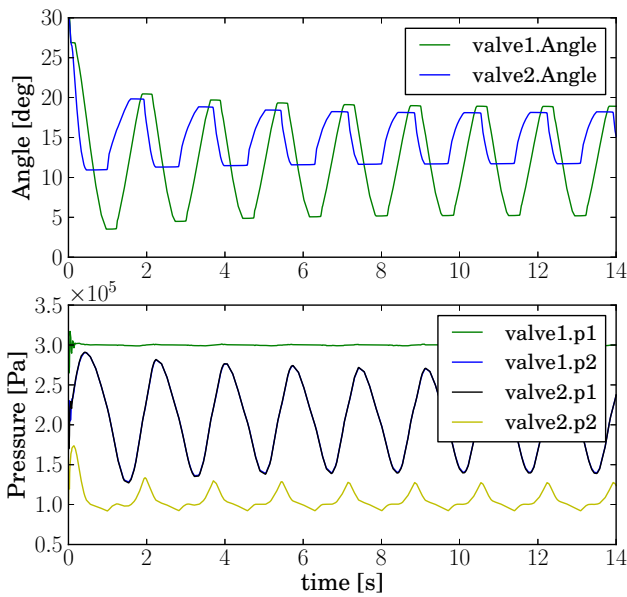


Figure 8. Results of oscillation test case

To demonstrate that the oscillations are caused by friction effects, the influence of friction and d’Alembert-forces was reduced with scaling parameters. A two-dimensional sweep of the quasi-steady-state amplitudes and periods over both scaling-parameters is shown in Figure 9.

It can easily be seen that the oscillations are strongly dependent on the friction forces and weakly dependent on the d’Alembert-forces. Furthermore, for vanishing friction-forces, the oscillations disappear completely. In other experiments, neglecting the d’Alembert-forces caused the oscillations to disappear, emphasising the importance of their inclusion in the model.

3.3 Dynamic interactions

The multi-domain nature of the presented model results in some interesting nonlinear transients. Two of them are presented in the following.

3.3.1 Aerodynamic Torque

The waterhammer effect is commonly known in pipeline operations. When a closing valve is used to stop the flow of a heavy and fast fluid-mass, the residual momentum of the fluid generates a build-up of pressure upstream of the valve.

For self-regulating pneumatic valves, a similar effect can occur: Let’s presuppose that the valve actuator closes the valve by a particular angle. The air mass upstream of the valve is then decelerated as a result, while generating a temporary pressure build-up. This pressure-buildup in turn increases the aerodynamic torque on the valve disk, closing the disk further and amplifying the effect.

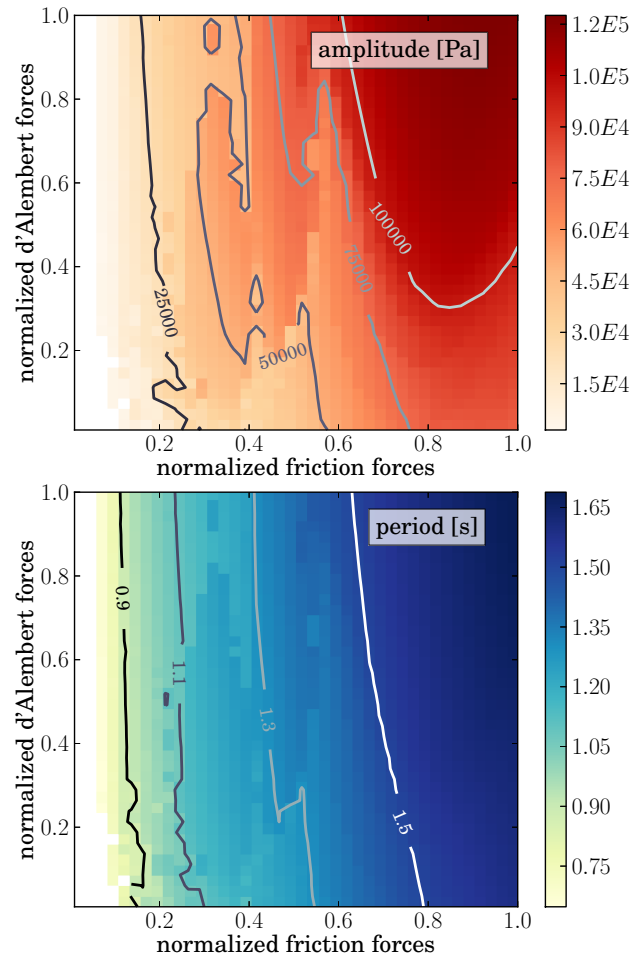


Figure 9. Results of oscillation test case

In Figure 10, a test model is represented where a pressure-regulated pipe is subjected to a harmonic inlet pressure with increasing frequency. The model was simulated with and without consideration of aerodynamic torque. The result of the simulation can be seen in Figure 11. It is easily recognizable that the valve opening is smaller when taking aerodynamic torque in consideration, especially at certain frequencies.

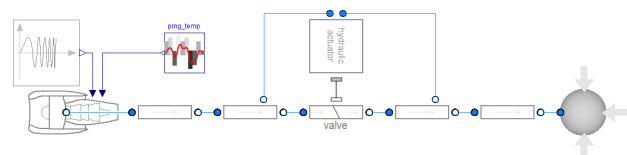


Figure 10. Aerodynamic Torque test model

3.3.2 Oscillatory heating

Generally, the environment of the valve has an ambient temperature different from the fluid temperature in the pipe. Also, heat conduction between environment and the valve chambers takes place. In the static case, the

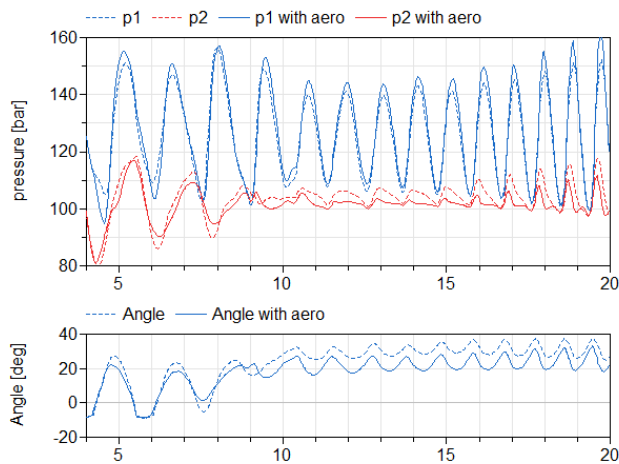


Figure 11. Transient effects of Aerodynamic Torque

temperature in the valve chamber will approach the ambient temperature after a time. However, in the case of valve movement, fluid mass is exchanged between the valve chambers and the pipe. In this way, the resulting temperature of the valve is dependent on the amount of valve movement.

4 Conclusion

Self-regulating pneumatic valves show a complex behavior, resulting in limit-cycle oscillations, if the overall system is not tuned satisfactorily. We present a detailed Modelica model for this kind of valves. The model includes all relevant physical effects, representing the thermal, fluid, and mechanical domains. Simulation results exhibit the typical dynamical characteristics of self-regulating pneumatic valves. Subsequently, the model can be used to predict system performance in an early development phase.

References

- Martin Aberger and Martin Otter. Modeling friction in modelica with the lund-grenoble friction model. In *Proceedings of the 2nd International Modelica Conference*, 2002.
- Peter Beater. Modeling and digital simulation of hydraulic systems in design and engineering education using modelica and hylib. In *Modelica workshop*, pages 23–24, 2000.
- Peter Beater and Christoph Clauß. Multidomain systems: Pneumatic, electronic and mechanical subsystems of a pneumatic drive modelled with modelica. In *Paper presented at the 3rd International Modelica Conference*, 2003.
- WL Bialowski. Dreams vs. reality: a view from both sides of the gap. *Pulp and Paper Canada*, 94:19–27, 1993.
- 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 *Proceedings of the Modelica Conference*, pages 631–640, 2006.
- MAA Shoukat Choudhury, Sirish L Shah, Nina F Thornhill, and David S Shook. Automatic detection and quantification of stiction in control valves. *Control Engineering Practice*, 14(12):1395–1412, 2006.
- C Canudas De Wit, Hans Olsson, Karl Johan Astrom, and Pablo Lischinsky. A new model for control of systems with friction. *Automatic Control, IEEE Transactions on*, 40(3): 419–425, 1995.
- Lane Desborough and Randy Miller. Increasing customer value of industrial control performance monitoring-honeywell’s experience. In *AICHE symposium series*, pages 169–189. New York; American Institute of Chemical Engineers; 1998, 2002.
- Martin Otter, Hilding Elmqvist, and Sven Erik Mattsson. The new modelica multibody library. In *Proceedings of the 3rd International Modelica Conference*. Citeseer, 2003.
- Sven-Erik Pohl and Markus Gräf. Dynamic simulation of a free-piston linear alternator in modelica. In *Modelica*, 2005.
- Aron Pujana-Arrese, Javier Arenas, Iban Retolaza, Ana Martinez-Esnaola, and Joseba Landaluze. Modelling in modelica of a pneumatic muscle: application to model an experimental set-up. In *21st European conference on modelling and simulation, ECMS*, pages 4–6, 2007.
- Michael Sielemann. *Device-Oriented Modeling and Simulation in Aircraft Energy Systems Design*. PhD thesis, Hamburg University of Technology, 2012a.
- Michael Sielemann. High-speed compressible flow and gas dynamics. In *Proceedings of the 9th International Modelica Conference*, 2012b.
- C Sollicc and F Danbon. Aerodynamic torque acting on a butterfly valve. comparison and choice of a torque coefficient. *Journal of fluids engineering*, 121(4):914–917, 1999.
- Michael Tiller. *Introduction to physical modeling with Modelica*. Springer Science & Business Media, 2001.

Dynamic Modeling of a Central Receiver CSP system in Modelica

Johan Edman¹ Johan Windahl²

¹Department of Energy Sciences, F. Eng., Lund University, Sweden, edman.jle@gmail.com

²Modelon AB, Ideon Science Park, Lund, Sweden, johan.windahl@modelon.com

Abstract

A dynamic model of the Solar Two test facility has been implemented in Modelica. The model consists of a set of Central Receiver specific CSP components, along with a Rankine cycle to form a complete system. Main components include models of a sun, heliostat field, receiver, storage tank and a Rankine cycle including a steam generator. The components and the full system were tested in a series of simulations – both dynamically and during steady state conditions – and the results were compared to data from the reference system. The dynamic behavior of the models aligned with expectations, although time constants could not be evaluated due to lack of dynamic reference data. The steady state characteristics were adequate for most models, although some complementary work needs to be done on the Receiver model.

Keywords: Modelica, Dymola, Dynamic modeling, Concentrated Solar Power, Central Receiver, Solar Salt, ThermalPower library

1 Introduction

Due to an increasing energy demand of a growing world population with an increasing consumption of technology, the interest renewable energy is ever increasing. This is further reinforced by a heightened awareness of the impact of the exploitation of non-renewable energy sources on local environments and global climate.

Furthermore, fossil fuels are increasingly being subjected to scrutiny. Reasons for this include the impending threat of global peak oil, i.e. the point when the extraction rate of oil can no longer meet the consumption rate, and the allocation of oil and other fossil resources to instable regions of the world. This has kindled the interest in developing new, local means of power production to minimize the dependence on foreign resources. A promising branch of renewable energy production is concentrated solar power (CSP).

Concentrated Solar Power refers to thermal power systems which use the sun as their primary heat source. The underlying principle behind CSP systems is that the solar radiation incident on Earth is basically collimated and

thus can be focused. To produce the heat flux required to maintain an adequate working temperature for efficient operation of a thermal power system the sunlight has to be concentrated several orders of magnitude.

There are several different types of CSP systems, but they all consist of the same elemental components. All CSP systems have a set of Sun-tracking reflectors which concentrate the sunlight onto an absorber. The absorbed power is converted into heat which is either converted directly into electricity using a heat engine or transported to a conventional thermal power cycle via a Heat Transfer Fluid (HTF). Systems using an HTF can often be combined with a heat storage system and/or an auxiliary gas turbine, enabling power generation during insufficient weather conditions and throughout the night.

The Central Receiver system (CRS), which is modelled in this project, uses numerous large mirrors, called Heliostats, which track the movement of the Sun over two axes. Heliostats focus the sunlight from a vast area onto a Receiver located on top of a tower. The HTF is heated in the receiver and then transported to the thermal power cycle, located by the foot of the tower. A sketch of the Solar Two system is presented in Figure 1.

This work is conducted in collaboration with Modelon AB, Sweden, which specializes in physical modeling of dynamic systems using Modelica. The models developed are largely based on the various model libraries in Modelons portfolio, especially the ThermalPower library, the LiquidCooling library and the Modelon Standard library.

This article describes the authors' process of modeling a CRS based on a reference system, and validating the

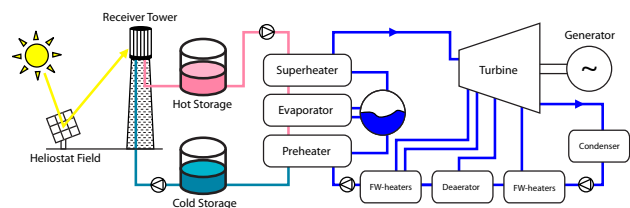


Figure 1. A sketch of the Solar Two CRS system.

model by conducting simulations. Section 2 describes the reference system, and the properties of the different components. In section 3 the equations governing the behavior of the components are stated, along with the property values used to configure the system model to match the reference system. Section 4 describes the simulations conducted to validate the models, and section 5 lists the simulation results. Finally, in section 6 a discussion of the results is given, along with conclusions and future work.

2 Reference system

A reference system was chosen to which the test results could be compared. It can be divided into two parts: The sun including weather conditions and the power plant.

2.1 Sun

Reference data for solar properties is taken from measured data collected at Planta Solar de Almeria, Spain (GeoModel Solar, 2014). Important properties are the direct normal insolation (DNI), which describes the solar flux intensity, the solar azimuth (γ_s) and elevation angle (α_s), which describe the solar position in the sky.

2.2 Power Plant

As most operating systems are commercially owned they do not publish much data. Therefore, an old government-funded test plant from the 1990's was chosen as reference system; the Solar Two project in California, USA. The Solar Two project was built to test and gain operating experience in using molten salt as HTF. It was an important predecessor of large, commercial plants such as the Gemasolar plant, formerly called Solar Tres, in Seville, Spain.

Reference data for Solar Two is taken from public reports (such as Pacheco, 2002) by Sandia National Laboratories, which was one of the main actors of the project.

Solar Two consists of a central tower in a surround heliostat field, an external cylinder receiver, a direct heat storage system and an electrical power generation system (EPGS) consisting of a steam generator producing steam for a 12.5 MW Rankine cycle. The HTF is a molten nitrate salt solution.

Heliostat Field The heliostat field is a north-biased surround field. It consist of 1818 relatively small heliostats à 39 m^2 and 108 larger heliostats à 95 m^2 , adding up to a total reflective area of $82,700 \text{ m}^2$.

Receiver The receiver consists of 24 panels arranged in a cylinder. Each panel contains 32 thin tubes through which the HTF flows. The exterior sides of the panels

are coated with black Pyromark paint, designed to have a high absorptivity (95%) and thermal endurance.

During normal operation the HTF enters at 290°C , and the flow is regulated so the exit temperature is kept constant at 565°C .

Storage System The direct heat storage system consists of two insulated storage tanks, one for hot HTF (565°C) and one for cold (290°C).

The salt flow through the receiver (cold tank \rightarrow hot tank) and the flow through the steam generator (hot tank \rightarrow cold tank) are independent of each other as long as none of the storage tanks are completely empty, and can be regulated separately.

Heat Transfer/Storage Medium The medium commonly referred to as Solar Salt is used both as heat transfer and as heat storage medium. It is a mixture of 60% sodium nitrate (NaNO_3) and 40% potassium nitrate (KNO_3). Solar salt has a high heat capacity and a low vapor pressure. It starts to crystallize at 240°C and is completely solid at 220°C (Ferri et al., 2008). Consequently, the salt has to be kept above these temperatures or it may cause major damage to parts of the system.

Steam Generator The steam generator consists of a preheater, a kettle evaporator and a superheater. The preheater and superheater are U-tube, straight shelled heat exchangers.

During normal operation, feedwater enter the preheater at 260°C and 100 bar . The preheater heat the feedwater to near a saturation temperature at 311°C , the evaporator produces steam at 311°C and finally the steam is heated in the superheater to 535°C .

The steam turbine, described in the next section, was salvaged from an old project and was not dimensioned to handle an inlet temperature of 535°C . Consequently, the steam had to be attemperated using feedwater to bring the temperature down to 510°C . The reason for producing steam at a higher temperature than the turbine could handle was to demonstrate the potential of the solar salt technology.

Rankine Cycle The power cycle is a non-reheat regenerative Rankine cycle using a train configuration with four extraction points. The first and second extraction points were fed to two high pressure feedwater heaters, the third point to a deaerator and the last point to a low pressure heater.

The cycle had a rated gross electrical output of 12.5 MW at 0.084 bar condenser pressure, a nominal steam mass flow rate of 13.9 kg/s and inlet conditions as described above, i.e. 510°C and 100 bar .

The steam turbine was salvaged from the Solar One project (the predecessor of Solar Two) which was conducted in the 1980's, and refurbished.

3 Implementation

Six major components, along with important subcomponents, were implemented: The Sun model, the Heliostat Field model, the Receiver model, the Storage Tank model, the Steam Generator model and the Rankine Cycle model. Components which are affected by ambient factors – such as ambient temperature, pressure and wind speed – contain an outer component called Weather Conditions, which allows these parameters to be set globally.

3.1 Sun

The Sun model provides input values to the Heliostat Field model. Outputs are DNI, elevation angle α_s and the azimuth angle γ_s . To calculate the elevation and azimuth angles the hour angle ω and the declination angle δ needs first to be calculated.

For a given point in time (day number n and solar time $hh:mm:ss$) and latitude λ the other angles are calculated as follows, here in degrees (Duffie et al., 2013):

$$\omega = 15(hh - 12) + mm/4 + ss/240 \quad (1)$$

$$\delta = 23.45 \sin(360(284 + n)/365) \quad (2)$$

$$\alpha_s = \sin^{-1}(\cos \delta \cos \omega \cos \lambda + \sin \delta \sin \lambda) \quad (3)$$

$$\gamma_s = \operatorname{sgn}(\omega) \left| \cos^{-1} \left(\frac{\cos \delta \cos \omega \sin \lambda - \sin \delta \cos \lambda}{\cos \alpha_s} \right) \right| \quad (4)$$

A simple correlation between solar altitude and DNI is adopted (Reno et al., 2012):

$$DNI = 950.2(1 - e^{-0.075\alpha_s}) [W/m^2] \quad (5)$$

3.2 Heliostat Field

The Heliostat Field model converts inputs from the Sun model into total insolation onto the receiver. The implementation can be summarized with the following formula:

$$I = DNI \cdot E_\eta(\alpha_s, \gamma_s) \cdot A \cdot \eta_r \cdot \eta_c \cdot \alpha_{rec} \quad (6)$$

where

$$\begin{aligned} I &= \text{insolation to receiver} && [W] \\ E_\eta(\alpha_s, \gamma_s) &= \text{Efficiency matrix} \\ A &= \text{Total reflective area} && [m^2] \end{aligned}$$

and

$$\begin{aligned} \eta_r &= \begin{cases} \text{reflectivity} & \text{if not inclRefl} \\ 1.0 & \text{else} \end{cases} \\ \eta_c &= \begin{cases} \text{cleanliness} & \text{if not inclClean} \\ 1.0 & \text{else} \end{cases} \\ \alpha_{rec} &= \begin{cases} 1/\text{rec. abs.} & \text{if inclRecAbs} \\ 1.0 & \text{else} \end{cases} \end{aligned}$$

The efficiency of the field is determined by an externally provided matrix. The efficiency matrix is inserted into a lookup table which takes the elevation and azimuth angles as input and outputs an interpolated efficiency value.

The following simplifications are used:

1. The entire heliostat field is always focused on the receiver, no capability to defocus the field is implemented.
2. The output is the total power reflected towards the receiver, no information on the distribution of the insolation from different directions or along the height or the receiver is provided.

3.3 Receiver

The receiver model converts incoming insolation into heat and transfers it to the HTF. It also accounts for ambient heat losses through radiation and convection. The implementation consists of three main subcomponents; a surface model, a wall model and a pipe model. Each of the subcomponents is discretized into n segments. The Receiver model is displayed in Figure 2.

A few simplifications have been made in this implementation:

1. The emissivity of the receiver surface is independent of the surface temperature (Gray body assumption).
2. The convective heat loss in each surface node is calculated as an estimate of the mean convective heat loss for a cylinder in cross flow with a surface temperature equal to the node temperature (see the Surface model section below).
3. Conduction heat losses are neglected.
4. Internal components used during start-up and shut-down sequences are left out for simplicity.

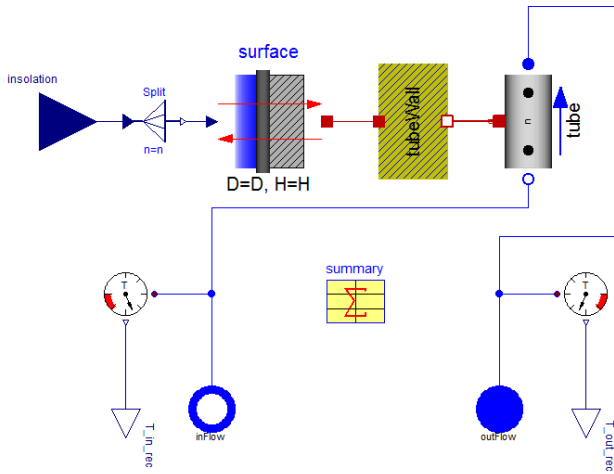


Figure 2. The Receiver model.

3.3.1 Surface model

The surface model is a subcomponent which calculates the heat exchanged between the receiver wall and its ambient surroundings. Insolation is converted to heat, heat losses through radiation and convection are subtracted and the net heat is transferred through the heat port.

The model is described by following equations:

$$\dot{Q}_{net} = \dot{Q}_{abs} - \dot{Q}_{rad} - \dot{Q}_{conv} \quad [W] \quad (7)$$

$$\dot{Q}_{abs} = \alpha I \quad [W] \quad (8)$$

$$\dot{Q}_{rad} = \varepsilon \sigma A (T_{wall}^4 - T_{sky}^4) \quad [W] \quad (9)$$

$$\dot{Q}_{conv} = h_c A (T_{wall} - T_{amb}) \quad [W] \quad (10)$$

where

$$I = \text{insolation} \quad [W]$$

$$\alpha = \text{surface absorptivity}$$

$$\varepsilon = \text{surface emissivity}$$

$$\sigma = \text{Stefan-Boltzmann constant} \quad [W/m^2 K^4]$$

$$A = \text{surface area} \quad [m^2]$$

$$h_c = \text{convective heat transfer coefficient} \quad [W/m^2 K]$$

The wall temperature T_{wall} is taken from the heat port, the ambient temperature T_{amb} is taken from the weather conditions model and the sky temperature T_{sky} is estimated as $T_{amb} - 8$ (Forristal, 2013).

The convective heat transfer coefficient h_c is calculated as the sum of forced and natural convection.

$$h_c = h_{cn} + h_{cf} \quad (11)$$

For the natural convective heat transfer coefficient, h_{cn} , a correlation for the Nusselt number for a vertical plate is used (Churchill et al., 1975). The formula is

modified to better fit vertical cylinders by adding a second term (Fujii et al., 1970).

$$h_{cn} = \frac{Nu_H k}{H} \quad (12)$$

$$Nu_H = \left(0.825 + \frac{0.387 (Gr_H Pr)^{1/6}}{\left(1 + \left(\frac{0.492}{Pr} \right)^{9/16} \right)^{8/27}} \right)^2 + 0.97 \frac{H}{D} \quad (13)$$

where

$$H = \text{height of the receiver} \quad [m]$$

$$D = \text{diameter of the receiver} \quad [m]$$

$$Gr_H = \text{Grashof number}$$

$$Pr = \text{Prandtl number}$$

For the forced convective heat transfer coefficient, h_{cf} , a correlation for the Nusselt number for a cylinder in cross-flow is used (Churchill et al., 1977):

$$h_{cf} = \frac{Nu_D k}{D} \quad (14)$$

$$Nu_D = 0.3 + \frac{0.62 Re^{1/2} Pr^{1/3}}{\left(1 + \left(\frac{0.4}{Pr} \right)^{2/3} \right)^{1/4}} \left(1 + \left(\frac{Re}{282000} \right)^{5/8} \right)^{4/5} \quad (15)$$

where

$$D = \text{diameter of the receiver} \quad [m]$$

$$Re = \text{Reynolds number}$$

3.4 Storage Tank

The storage tank model consists of a two-media volume, a wall and two heat conductors. Heat ports of the volume are connected to a fixed temperature source via the wall model and two conductor models (see the Film Conductor section below). Values for wall properties are given as the lumped properties of the metal wall and the insulation.

3.4.1 Film Conductor

The FilmConductor model is created to provide a conductor model with a variable heat transfer area. The heat transfer area is set to depend on the liquid level in the tank. As the conductor model is separate from the wall model, the conduction through the liquid layer closest to the tank wall is modelled. The thermal conductivity is evaluated at the film temperature. It is described by following equations:

$$\dot{Q} = k(T_{film}) \cdot A_{heat} \cdot (T_{fluid} - T_{wall}) \quad (16)$$

$$T_{film} = \frac{T_{wall} + T_{fluid}}{2} \quad (17)$$

where

$$k = \text{Thermal conductivity} \quad [W/m K]$$

The thickness of the film is given a default value of 0.2 m.

3.5 Steam Generator

In the steam generator model, heat from the HTF is transferred to the water in the Rankine cycle in three static HEX models; the preheater, the boiler and the superheater. Steam is produced in a drum model connected to the boiler. The feedwater flow is regulated to keep the drum level constant. Volumes are placed between the flow components to provide numerical stability and thermal inertia. The Steam Generator model is presented in Figure 3.

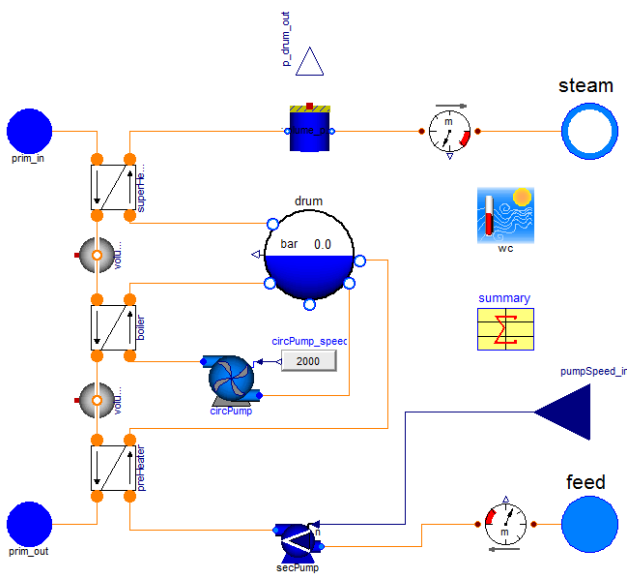


Figure 3. The Steam Generator model.

3.6 Rankine Cycle

The Rankine cycle model is a slightly simplified model of the Solar Two steam cycle plant. It consists of four turbine segment, a condenser, a generator (modelled as an efficiency parameter) and three open feedwater heater components. The turbine models are based on Stodola's law. The heater components consist of a deaerator, a pump and a check valve. The pump speed is regulated to keep the liquid level in the deaerator at a constant level.

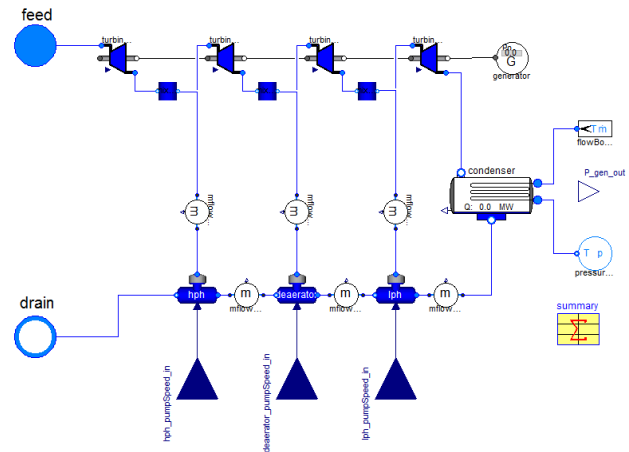


Figure 4. The Rankine model.

Preheaters and turbine segments are connected in a train configuration. Only open feedwater heaters (deaerators) are used as it allows for easy configuration of the setup (Haywood, 1991). The Rankine model is displayed in Figure 4.

3.7 Configuration of the System model

In the system model, components are combined and configured according to data from the reference system presented in section 2.2. All salt/water flows are regulated by a master control model.

As no efficiency matrix has been published for the Solar Two heliostat field, a matrix calculated for a similar type of field was modified and used. The matrix is generated by Sandia National Laboratories using an algorithm called DELSOL, written in FORTRAN (Ehrhart et al., 2013). Inputs to DELSOL differ slightly from properties of the Solar Two plant, which will cause an error. However, in the absence of a better approximation this matrix was used.

The heat transfer properties of the storage tanks were set to guess values. No effort was made to fine tune the heat loss from the tanks as its magnitude was several orders smaller than the heat transported through the tanks.

The Rankine model is configured to maximize its efficiency. The efficiency of a non-reheat, regenerative Rankine cycle with only open feedwater heaters is maximized when the enthalpy rise is equal between two adjacent heaters (Haywood, 1991). Knowing the condenser back pressure and final feedwater temperature, the enthalpy in the condenser and high pressure heater can be computed. The intermediate enthalpy levels can then be calculated and the pressures are determined as the saturation pressures with corresponding liquid phase enthalpies.

Setting the isentropic efficiency of each turbine segment to 0.7, which is reasonable for a turbine of this size

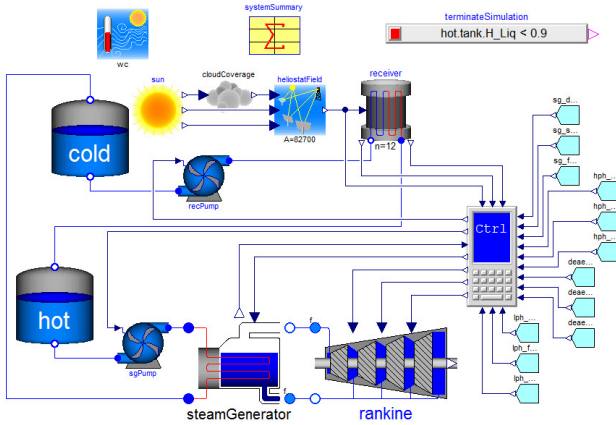


Figure 5. The System model setup.

and age (Haywood, 1991; Thern, 2013), the enthalpy levels in the turbine are iteratively calculated as

$$h_{i+1} = h_i - \eta_{is}(h_i - h_{is}(p_{i+1})) \quad (18)$$

starting with the entalpy of the inlet steam.

The final values of pressures and entahlpies at the nominal operating point are presented in Table 1.

Parameter	Pressure [bar]	H, steam [kJ/kg]	H, liquid [kJ/kg]
Inlet steam	100.000	3464.28	
1st extraction	46.920	3294.26	1134.83
2nd extraction	13.065	3051.71	815.80
3rd extraction	1.885	2769.13	496.76
Condenser	0.084	2435.92	177.73

Table 1. Nominal pressure and enthalpy values for the Rankine model.

The full system model is displayed in Figure 5.

4 Testing and Simulation

Each component was tested dynamically and in steady state and results were compared to data from the reference system if it was available. In this section setups from the Receiver and the full system tests are presented. For a further detailed description of the tests, see Edman (2014).

4.1 Receiver

Two tests are presented for the receiver; one test of the dynamics and one test measuring the efficiency at steady state conditions.

4.1.1 Dynamics

The receiver is initially fed a constant insolation of 35 MW and a salt mass flow rate of 60 kg/s. At

$t = 5$ min, the insolation is increased to 45 MW, and at $t = 10$ min the mass flow rate is increased to 80 kg/s. The temperature of the salt outflow and in the tubewalls, and the total heat loss at the surface are monitored.

4.1.2 Steady State

The steady state efficiency of the receiver was tested at different levels of insolation and wind speed. The salt flow was regulated to keep the receiver outflow temperature constant at 565°C. Once the flow is stabilized the efficiency is calculated as:

$$\eta_{rec} = \frac{Q_{net}}{Q_{inc}} = \frac{\dot{m} \cdot (h_{out} - h_{in})}{I} \quad (19)$$

The input values are presented in Table 2.

Parameter	Values	
Insolation	{48, 40, 30, 20, 15}	[MW]
Wind speed at 76.2 m	{0, 2.5, 5, 7.5, 10}	[m/s]

Table 2. Parameter values used in the Receiver Steady State test.

4.2 System

Two tests are conducted including all parts of the system, one steady state test and one dynamics test. Parameter and initial values are configured according to section 2.2.

4.2.1 Steady State

In the steady state test the receiver is fed a constant level of insolation. The mass flow rate of salt through the receiver is regulated to keep the outlet temperature at a constant level of 565°C. The steam generator is fed with the same salt flow rate as the receiver, thus keeping the salt level in the tanks constant. Once the system has reached steady state, heat flow rates and efficiencies are determined and compared to reference data (Pacheco, 2002).

Four simulations are run with diffent levels of insolation. The input parameters are listed in Table 3.

	Case 1	Case 2	Case 3	Case 4
Time	12:00	12:00	09:00	09:00
Day	172	354	172	354

Table 3. Times for the System Steady State test.

4.2.2 Dynamics

The whole system is simulated over a couple of full day scenarios, Clear and Cloudy. Salt flows are regulated to

maintain nominal operating conditions. Each simulation is started at $t = 06:00$ as the start-up sequence of the receiver is not modelled. The simulation is terminated automatically when the salt level in the hot tank drops below 0.9 m .

Both simulations are run on day number 172. In the Clear scenario a clear day is simulated. In the Cloudy scenario a cloud appears at $t = 11:00$ and obstructs the insolation for one hour. The energy flow rates in the different parts of the system as well as the tank levels are monitored.

5 Results

5.1 Receiver Dynamics

Results from the dynamic receiver test are presented in Figure 6. In the first 100 s the receiver tube walls are heated from their initial value of 290°C to their steady state value.

When the insolation is increased at $t = 5\text{ min}$ the wall temperature starts to rise, leading to an increased heat flow rate to the salt flow and thus an increased outflow temperature. When the salt flow rate is increased at $t = 10\text{ min}$ the outflow temperature drops as a greater amount of salt is heated. The increasing temperature difference along with the increased flow rate leads to an increased heat flow rate to the salt flow, causing the wall temperature to drop.

The rise times of the slopes are approximately equal to the time it takes for one mass unit of salt to pass through the receiver ($\approx 60\text{s}@60\text{kg/s}$ and $45\text{s}@80\text{kg/s}$). The smoothing at the end of the slopes is due to the thermal inertia of the tube walls.

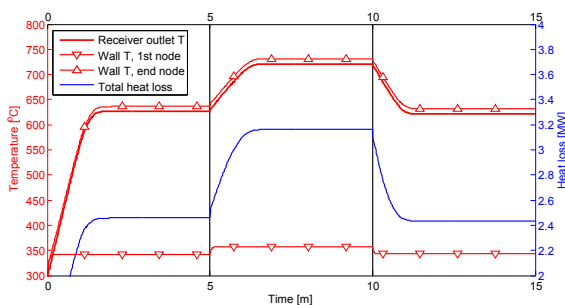


Figure 6. The results from the Receiver dynamics test. Color indicates the correct y-axis.

5.2 Receiver Steady State

Receiver efficiencies at steady state conditions for different levels of insolation and wind speed are presented in Figure 7.

The efficiency peaks at high insolation and low wind speed and is steadily decreasing with decreasing insola-

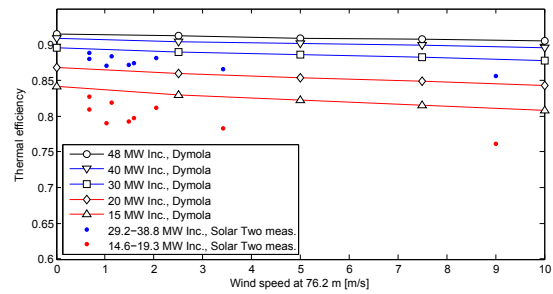


Figure 7. Steady state efficiency of the Receiver model at different levels of insolation and wind speed. The colored markers indicate measured values from the Solar Two project.

tion and increasing wind speed. The blue and red markers in the figure are measured values from the Solar Two project (Pacheco, 2002). Although the behavior of the receiver model is correct the efficiency of the model is generally higher than the measured values from the Solar Two project.

5.3 System Steady State

A visual presentation of the first steady state case is presented in Figure 8. The majority of the losses in the system occur in the heliostat field (optical losses) and in the condenser of the Rankine cycle, and thus these are interesting areas to analyze when developing the system as small improvements may have large impacts.

The power output is higher than the rated power of the Solar Two Turbine (12.5 MW) as the steam is not attemperated. Also, the salt flow is higher than the nominal flow as it regulated by the receiver flow. If the steam generator salt flow was separately regulated, this would correspond to the hot tank salt level rising.

A comparison between the solar specific component efficiencies in the four different cases and measured values from the Solar Two project is presented in Table 4.

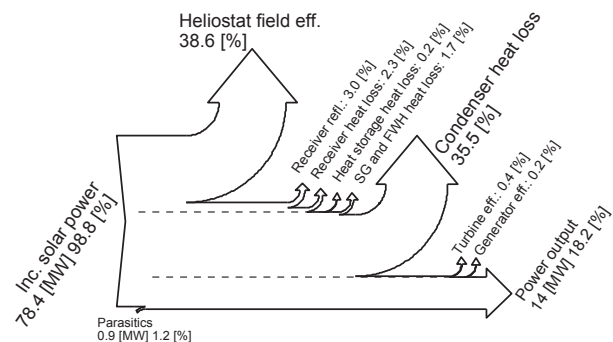


Figure 8. Visualization of the losses from different parts of the system in the System Steady State test. Solid lines are the cloudy scenario and the dashed lines are the clear scenario

	Efficiency				
HS field	Case 1	Case 2	Case 3	Case 4	Solar Two
Receiver	0.71	0.65	0.67	0.54	0.63
Overall	0.91	0.91	0.91	0.89	0.88
	0.18	0.17	0.17	0.13	0.13

Table 4. Comparison between simulated efficiencies and the measured Solar Two efficiencies(Pacheco, 2002).

The heliostat field and receiver models are in general more efficient than the measured values, resulting in a generally higher overall efficiency. The efficiencies of the other parts of the system did not differ significantly from the measured values.

5.4 System Dynamics

Figure 9 shows energy flow rates in different parts of the system (top), and the liquid levels in the tanks (bottom), from both scenarios.

Power production continues virtually undisturbed during the passing of the cloud and well after the insolation sinks below the power input demanded by the power cycle. However, in the cloudy scenario the simulation is stopped earlier as power has been drained from the storage during the passing of the cloud and thus the hot tank empties sooner.

The level in the hot tank sinks in the early hours as the absorbed power by the receiver is lower than the power delivered to the steam generator. Once the absorbed power exceeds the demand the level starts to rise, and continues to do so until the demand is once again higher than the absorbed power. In the Cloudy scenario

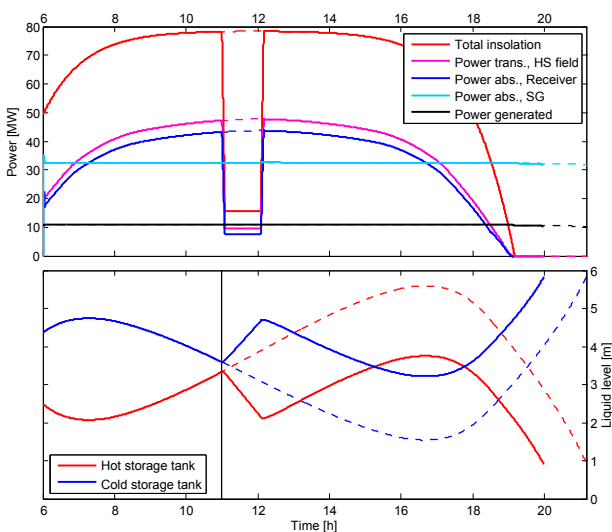


Figure 9. Top: Power transferred in the different parts of the system in the System Dynamics test. Bottom: Variations in tank liquid levels. The dashed lines represent the Clear scenario and the solid lines represent the Cloudy scenario.

the hot salt level drops at a constant rate during the passing of the cloud as the incoming flow from the receiver is very low.

6 Discussion

As relevant dynamic data from the reference system is scarce, time constants and general appearance of the dynamic results cannot be validated against a reference, only discussed from a theoretical point of view. To get feedback from the reference system the steady state heat loss and efficiency tests were conducted.

6.1 Evaluation of Results

6.1.1 Receiver model

The dynamical behavior of the receiver is satisfactory. However, there is room for improvement as start-up and shut-down procedures are not modeled.

The steady state efficiency of the receiver model is generally higher than the measured values. A few influential factors can be mentioned:

1. No conduction heat loss has been implemented in the receiver model, see further discussion below.
2. The insolation is always uniformly distributed among the surface nodes in the model.
3. In the Solar Two receiver efficiency measurement, neither inlet nor outlet salt temperatures were stringently kept at their nominal values (Pacheco, 2002).

Conduction is seldom mentioned in conjunction with receiver heat loss and it was assumed that its influence is negligible. This assumption may need to be reevaluated. The difference between simulated and measured efficiency seems to be proportional to the level of insolation, suggesting a constant heat loss factor is missing from the model. This factor could be due to conduction or possible an inaccurate absorptivity of the receiver surface. However, it does not appear to vary significantly with wind speed, which suggests that the convective heat loss is properly modelled.

6.1.2 System model

Steady State The overall efficiency of the system model is generally higher than the measured values. Differing factors are mainly the heliostat field, the receiver (already discussed) and parasitics.

The efficiency of the heliostat field model is higher than the Solar Two HS field. As this efficiency is determined by the efficiency matrix, using a correct matrix would eliminate this error. As the heliostat field is one of the major power sinks in the system, a correct description of its properties is important.

Parasitics have not been thoroughly studied in this project and many of the parasitic components of the real system have been omitted from the models. Therefore, the parasitic efficiency was higher in the model.

Dynamics The dynamic behavior of the system aligned with expectations and demonstrates the benefit of using a direct storage system. The power production is virtually undisturbed by the passing of the cloud. It is clear that the capacity of this particular storage system is not very large, as the liquid level sinks quickly when the incoming flow of hot salt is disrupted. A commercial system with a larger storage capacity would be able to handle even more perturbances. However, during particularly bad conditions an auxiliary heat source would be needed.

6.2 Conclusions

The dynamic behavior of the receiver model is properly modelled to what extent it is possible to verify. Responses to sudden changes in the input parameters are gradual and the model stabilizes at reasonable values. However, it is hard to evaluate time constants of the system as most data from the reference system is given as efficiencies and average values, and dynamic data is very scarce.

The two components which are unique for the CRS system are the heliostat field and the receiver. The heliostat field model utilizes the data which is most commonly given for a real field, but more detailed data is needed to improve the model. The receiver model is more efficient than the real system, which is most likely due to the lack of conduction losses and possibly a faulty absorptivity value. More detailed inputs from the heliostat field and the ambient conditions would also improve the model.

The EPGS models work properly, which validates the usefulness of the Modelon ThermalPower library as most components are taken from there.

All models are generic and rescalable. However, the Rankine model has to be modified if a different configuration is used, e.g. with more extraction points; with closed feedwater heaters or with a reheat configuration.

Dymola and the Modelica language, are powerful tools for modeling the thermohydraulic parts of the system, i.e. the solar loop and the power cycle. For detailed modeling of complex optical systems such as the heliostat field, an optical simulation tool would be needed.

6.3 Future work

More detail could be added to increase accuracy and allow for more specific properties to be studied.

1. Heliostat/Receiver models: More detailed information about the insolation patterns onto the receiver are needed to increase the model accuracy.
2. HTF model: To be able to model filling and draining of receiver and pipes. During start-up and shut-down sequences the HTF medium must be able to handle a mixture of solar salt and air.
3. Storage Tank model: Stratification and auxiliary heating of the storage tanks have not been modelled.
4. Steam Generator model: Steam attemperation with feedwater should be included in the steam generator model. Also, changing the static heat exchanger models to dynamic ones should be considered.
5. Rankine model: A more generic Rankine model could be implemented.

References

- W. Churchill, H. H. S. Chu, "Correlating equations for laminar and turbulent free convection from vertical plate," *Int. J. Heat. Mass. Tran.* 18:1323-1329, 1975.
- S. W. Churchill and M. Bernstein, "A Correlating Equation for Forced Convection From Gases and Liquids to a Circular Cylinder in Crossflow," *Int. J. Heat. Mass. Tran. Trans. ASME* 99, 1977, pp. 300-306.
- J. A. Duffie and W. A. Beckman, "Solar Radiation" in *Solar Engineering of Thermal Processes*, 4th. Ed., Hoboken, New Jersey: Wiley, 2013, pp. 12-20.
- J. Edman, "Dynamic Modeling of a Central Receiver CSP sytem in Dymola," M.S. thesis, Dept. En. Sci., Lund Univ., Lund, Sweden, 2014.
- B. D. Ehrhart and D. D. Gill, "Evaluation of Annual Efficiencies of High Temperature Central Receiver Concentrated Solar Power Plants With Thermal Energy Storage," Sandia Nat. Lab., Albuquerque, NM, Rep. SAND2013-5493, Jul. 2013.
- R. Ferri, A. Cammi and D. Mazzei, "Molten salt mixture properties in RELAP5 code for thermodynamic solar applications," *Int. J. Therm. Sci.* vol 47, 2008, pp. 1676-1687.
- R. Forristal, "Heat Transfer Analysis and Modeling of a Parabolic Trough Solar Receiver Implemented in Engineering Equation Solver," NREL, Golden, CO, Rep. NREL/TP-550-34169, Oct. 2013.
- T. Fujii, H. Uehara, "Laminar natural-convective heat transfer from the outer surface of a vertical cylinder," *Int. J. Heat. Mass. Tran.* 13:607-615, 1970.
- GeoModel Solar, Typical Meteorological Year Data (Sample Data), GeoModel Solar, [Online], Available: <http://geomodelsolar.eu/data/typical-meteorological-year>, [Accessed: Feb. 2014].

- R. W. Haywood, "Advanced steam-turbine plant" in *Analysis of Engineering Cycles*, 4th. Ed., Pergamon Press, 1991, pp. 110-115.
- J. E. Pacheco, "Final Test and Evaluation Results from the Solar Two Project," Sandia Nat. Lab., Albuquerque, NM, Rep. SAND2002-0120, Jan. 2002.
- M. J. Reno, C. W. Hansen and J. S. Stein, "Global Horizontal Irradiance Clear Sky Models: Implementation and Analysis," Sandia Nat. Lab., Albuquerque, NM, Rep. SAND2012-2389, Mar. 2012.
- M. Thern, Lund University, F. Eng., Dept. En. Sci., private communication, Dec 2013.

Modeling of Linear Concentrating Solar Power using Direct Steam Generation with Parabolic-Trough

Antoine Arousseau^{1,2} Valéry Vuillerme¹ Jean-Jacques Bezian²

¹ Univ. Grenoble Alpes, INES, F-33375 Le Bourget du Lac, France

CEA, LITEN, Laboratoire des Systèmes Solaires Haute Température, antoine.rousseau@cea.fr;

² Université de Toulouse, Mines Albi, CNRS, Centre RAPSODEE, France;

Abstract

This paper deals with the Modelica /Dymola modeling of linear concentrating solar power in a parabolic-trough experimental loop using direct steam generation. An extensive description of the parabolic collector and the absorber tube models is proposed. First results of the simulation of a clear sky day, with the aim of validating the models, are discussed. Experimental data from the CIEMAT-PSA DISS loop in Almeria, Spain, is used.

Keywords: Concentrating Solar Power, Parabolic-Trough, Direct Steam Generation, Modeling, ThermoSysPro.

1 Introduction

Concentrating Solar Power (CSP), or Solar Thermal Electricity, is a promising technology for renewable electricity generation. In its latest Technology Roadmap report (OECD/IEA, 2014), the International Energy Agency estimates that with appropriate R&D support, the contribution of CSP to the global electricity production could reach 11% by 2050.

Among the several CSP technologies, parabolic-

focal line of a parabolic mirror. The process of using water as the heat transfer fluid in the tubes and generating steam for a direct use as the working fluid of a thermodynamic cycle is referred as Direct Steam Generation (DSG). It offers several advantages and has potential cost reduction effects, compared to technologies using other heat transfer fluids and heat exchangers (Eck et al., 2008; Feldhoff, Eck, Benitez, & Riffelmann, 2009).

The combination of the natural transient condition of solar irradiation and the dynamics induced by the presence of a two-phase flow inside the absorber tubes results in a behavior of the steam generation system that is strongly dynamic. Modeling this behavior at the system scale is useful for the sizing and design of both the solar field and its control system.

This paper presents a model of a parabolic-trough solar field, developed with Modelica on the basis of the ThermoSysPro library, developed by EDF R&D (ThermoSysPro 2014). In the first section, the Modelica model is presented, with a focus on the parabolic collectors and the absorber tubes, and the second section presents the preliminary simulations carried out to validate the models using the

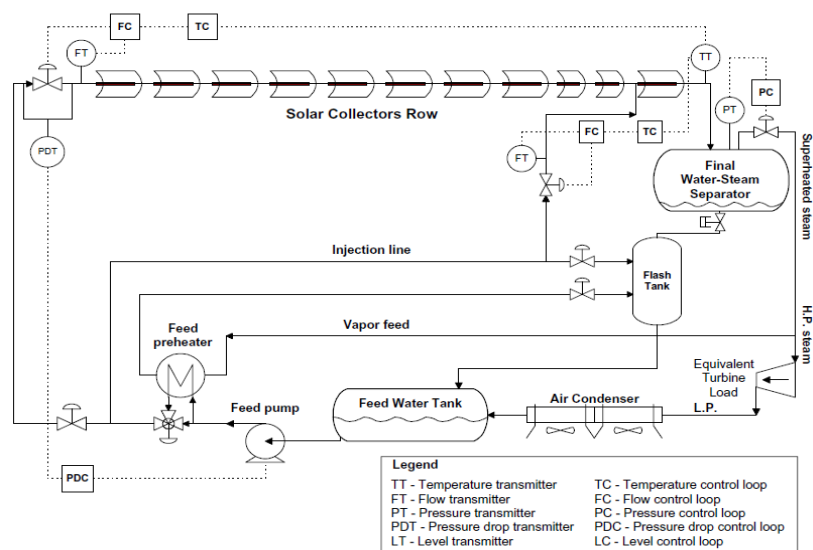


Figure 1: Diagram of the DISS loop (Valenzuela et al. 2005)

trough uses linear concentration to collect heat with a fluid flowing inside an absorber tube located at the

experimental data of the CIEMAT-PSA DISS

experimental loop in Almeria, Spain. Simulations are carried out with the commercial software Dymola.

2 Models description

2.1 The reference experimental setting

The DISS (for Direct Solar Steam) experimental loop is located in Almeria, Spain, and is operated by the CIEMAT-PSA institute. It has been operated since about fifteen years, and many studies have been published. It consists in the connection in series of several parabolic-trough collectors, and the appropriate balance of plant installations for the water/steam flow. This study is making use of the experimental loop operated in “once-through” mode, where water is vaporized and steam superheated in the same absorber line, without separation. Description of the experimental setting and collectors details can be found in (Valenzuela, Zarza, Berenguel, & Camacho, 2004, 2005). Figure 1 shows the experimental loop for the once-through operation mode. It here consists in 11 collectors connected in series, with an injection cooler between the 10th and 11th collector for the control of the outlet steam temperature. Two collectors are 25 meters long and the other ones are 50 meters long.

2.2 Model structure

As only the solar field section is modeled (consisting of the 11 connected collectors), presented here is the general structure of a single collector model, consisting of a parabolic mirror and an absorber tube. Figure 2 pictures the structure.

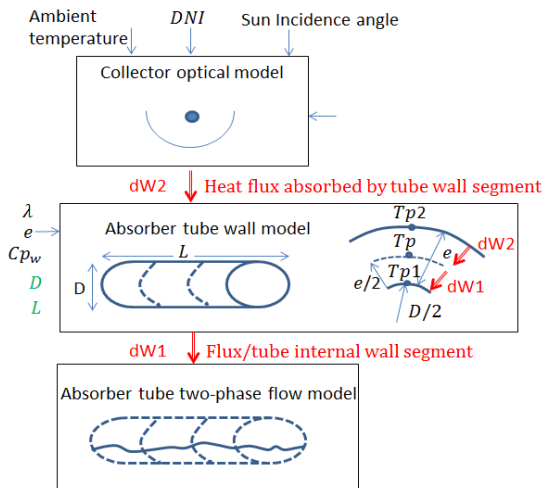


Figure 2: Collector-tube model structure

The optical model computes the heat flux absorbed by the tube wall, then the tube wall model computes the flux through the wall, and the tube two-phase flow model eventually computes the flow conditions. The

three “sub-models” are connected with thermal ports and exchange heat flux and temperature data.

2.3 Collector model

A LS3-type collector is modeled.

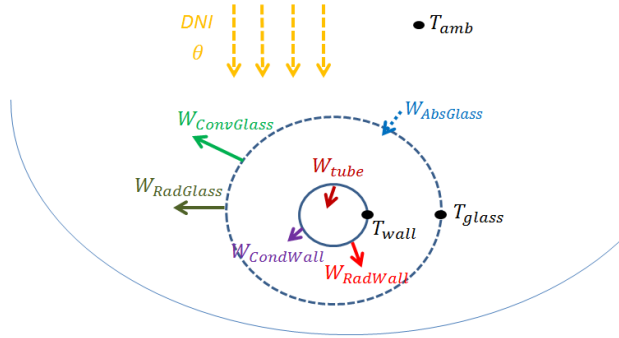


Figure 3: Heat flux diagram on the collector

Figure 3 pictures the collector model in terms of heat fluxes. A developed modified version of the ThermoSysPro 3.1 solar collector is used. The absorber tube and the parabolic mirror are discretized into a defined number of segments, with a set of acausal equations for each of them. The heat flux absorbed by a tube internal wall segment is computed with the following equation:

$$-W_{tube} = \eta_{Opt,over} \times IAM \times \cos \theta \times DNI \quad (1)$$

$$\times \frac{A_{refl}}{N_s} - W_{RadGlass}$$

$$- W_{ConvGlass}$$

The sign of the heat flux is negative from the parabolic collector point of view, since flux leaving a component is negative by convention. The glass envelope energy balance is computed by the following equation:

$$d_M C_{Pglass} \frac{dT_{glass}}{dt} \quad (2)$$

$$= W_{AbsGlass} + W_{CondWall}$$

$$+ W_{RadWall} - W_{ConvGlass}$$

$$- W_{RadGlass}$$

Following equations compute the other heat flux terms:

$$W_{RadWall} = \frac{A_{tube}}{N_s} \times \sigma \times \varepsilon_{tube} \times (T_{wall}^4 \quad (3)$$

$$- T_{glass}^4)$$

$$W_{CondWall} = \frac{A_{Tube}}{N_s} \times \lambda \times (T_{wall} - T_{glass}) / \frac{D_{tube}}{2 \log \frac{D_{glass}}{D_{tube}}} \quad (4)$$

$$W_{RadGlass} = \frac{A_{glass}}{N_s} \times \sigma \times \epsilon_{glass} \times (T_{glass}^4 - T_{sky}^4) \quad (5)$$

$$W_{ConvGlass} = \frac{A_{glass}}{N_s} \times h \times (T_{glass} - T_{amb}) \quad (6)$$

$$W_{AbsGlass} = DNI \times \frac{A_{glass}}{N_s} \times \alpha_{glass} \times \cos \theta \times IAM \times \eta_{Opt,Peak} \quad (7)$$

The equations terms are detailed in Table 1.

W_{tube}	Heat flux transmitted by tube wall
$W_{RadGlass}$	Heat flux loss through radiation of glass envelope to atmosphere
$W_{ConvGlass}$	Heat flux loss through convection of glass envelope to atmosphere
$W_{AbsGlass}$	Heat flux absorbed by glass envelope
$W_{CondWall}$	Conduction heat flux from tube wall to glass envelope
$W_{RadWall}$	Radiation heat flux from tube wall to glass envelope
$\eta_{Opt,over}$	Overall (glass and tube) collector efficiency
IAM	Incidence angle modifier
θ	Incidence angle
DNI	Direct Normal Irradiation
A_{refl}	Parabolic mirror aperture area
N_s	Number of discretization segments
d_M	Mass of glass envelope segment
$C_{P,glass}$	Glass envelope thermal capacity
T_{glass}	Glass envelope temperature
A_{tube}	Tube wall heat exchange area
σ	Boltzmann constant
ϵ_{tube}	Tube wall emissivity
T_{wall}	Tube wall temperature
λ	Inner gas conductivity
D_{tube}	Tube diameter
D_{glass}	Glass envelope diameter
T_{sky}	Sky temperature
h	Convection heat loss coefficient
T_{amb}	Ambient external temperature
α_{glass}	Glass absorptivity at normal incidence
$\eta_{Opt,Peak}$	Peak parabolic mirror optical efficiency

Table 1 : Collector model terms detail

The incidence angle modifier is a function of the incidence angle and is extracted from (Valenzuela et al., 2005) :

$$IAM = 1 - 0.00188 \times \theta - 0.000149206 \times \theta^2 \quad (8)$$

2.4 Two-phase flow model

A developed modified version of the dynamic two-phase flow tube model of the ThermoSysPro 3.1 library is used. Pressure drop correlations were modified from the original version. The two-phase flow tube model is connected to the tube wall model through a thermal port and to other fluid components through fluid ports. The ThermoSysPro structure model and the two-phase flow tube model (highlighted in a circle) are pictured on Figure 4.

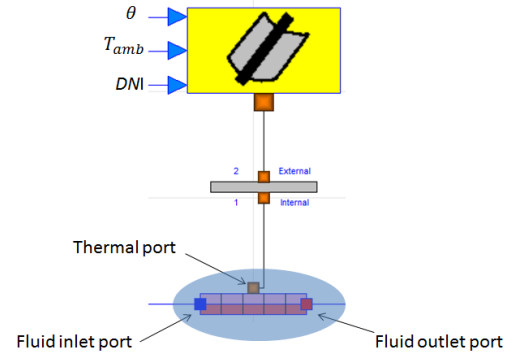


Figure 4: ThermoSysPro collector and tube model diagram

Tubes are discretized only in the longitudinal direction, since ratio between length and diameter is very large. Pressure P and specific enthalpy h are state variables. Mass, energy, and momentum conservation equations, for each i segment of cross section area A yield:

$$A dx \left(\frac{\partial \rho}{\partial h_{[i]}} \frac{\partial h}{\partial t_{[i+1]}} + \frac{\partial \rho}{\partial P_{[i]}} \frac{\partial P}{\partial t_{[i+1]}} \right) = Q_{[i]} - Q_{[i+1]} \quad (8)$$

$$A dx \left[\left(h_{i+1} \frac{\partial \rho}{\partial P_{[i]}} - 1 \right) \frac{\partial P}{\partial t_{[i+1]}} + \left(h_{i+1} \frac{\partial \rho}{\partial h_{[i]}} + \rho_{[i]} \right) \frac{\partial h}{\partial t_{[i+1]}} \right] = h_{b[i]} Q_{[i]} - h_{b[i+1]} Q_{[i+1]} + dW_{[i]} \quad (9)$$

$$\frac{1}{A} \frac{\partial Q}{\partial t_{[i]}} dx = P_{[i]} - P_{[i+1]} - dpf_{[i]} - dp g_{[i]} - dpa_{[i]} \quad (10)$$

With the density ρ , pressure P , mass flow rate Q , cell boundary specific enthalpy h_b , exchanged thermal power dW , friction pressure loss dpf , gravity pressure loss $dp g$, acceleration pressure loss dpa . Dynamics terms in equation (10), like the acceleration pressure term or the inertia term (left hand side) can be set to zero for computations without a full dynamic

modeling. A Staggered grid is used for the spatial discretization, with the momentum balance equation (10) computed at control volume boundaries.

2.4.1 Closure equations: pressure losses

The gravity pressure loss and the acceleration pressure loss terms are computed with homogeneous flow assumptions. The friction pressure loss is computed using separate flows assumption and the Martinelli-Nelson method: In two-phase flow regions, friction pressure loss is computed as the product of the liquid only-pressure loss and a two-phase flow multiplier:

$$dpf_{2\phi} = \Phi_{LO}^2 \times dpf_{LO} \quad (11)$$

dpf_{LO} is computed as the friction pressure drop with only liquid flowing at full rate, using classical equations. The multiplier Φ_{LO} is computed using the Friedel empirical correlation, which was implemented in the model and is considered as the best correlation for this range of mass fluxes:

$$\Phi_{LO}^2 = E + 3.24 \times F \times H \times Fr^{-0.045} We^{-0.035} \quad (12)$$

with

$$E = (1-x)^2 + x^2 \frac{\rho_l f_{LO}}{\rho_g f_{GO}} \quad (13)$$

$$F = x^{0.78} (1-x)^{0.224} \quad (14)$$

$$H = \left(\frac{\rho_l}{\rho_g}\right)^{0.91} \left(\frac{\mu_g}{\mu_l}\right)^{0.19} \left(1 - \frac{\mu_g}{\mu_l}\right)^{0.7} \quad (15)$$

$$Fr = \frac{Q^2}{A^2 \bar{\rho}^2 g D_{tube}} \quad (16)$$

$$We = \frac{Q^2 D_{tube}}{A^2 \bar{\rho} \sigma_s} \quad (17)$$

With x the steam fraction, ρ_l the liquid water density, ρ_g the steam density.

Liquid-only and steam-only friction coefficients are computed using classical equations involving liquid-only and steam-only Reynolds numbers:

$$f_{LO} = \frac{0.079}{Re_{LO}^{0.25}} \quad (18)$$

$$f_{GO} = \frac{0.079}{Re_{GO}^{0.25}} \quad (19)$$

μ_l and μ_g are the water and steam densities, σ_s the surface tension. Fr and We are the Froude and Weber dimensionless numbers. The average density $\bar{\rho}$ is computed the following way:

$$\bar{\rho} = \left(\frac{x}{\rho_g} + \frac{1-x}{\rho_l}\right)^{-1} \quad (20)$$

2.4.2 Closure equations: heat transfer coefficient

For each tube segment, the absorbed heat flux is computed with the tube inner wall temperature T_p and the fluid segment temperature T_1 :

$$dW = h \times dS \times (T_p - T_1) \quad (21)$$

The heat transfer coefficient in single-phase flow region is computed using Dittus-Boelter equation:

$$h = 0.023 \frac{k}{D_{tube}} Re^{0.8} Pr^{0.4} \quad (22)$$

With k thermal conductivity. In two-phase flow region, the heat transfer coefficient is computed using the superposition method of the Chan correlation, described in (Odeh, Morrison, & Behnia, 1998) :

$$h_{2\phi} = E h_{cl} + S h_{eb} \quad (23)$$

The single-phase convective boiling term h_{cl} is computed with the Dittus-Boelter equation (22). E is its related corrective term and is computed with a correlation to the Martinelli parameter and the boiling number BO :

$$E = 1 + 24000 BO^{1.16} + 1.37 X_{tt}^{-0.86} \quad (24)$$

With X_{tt} the Martinelli parameter:

$$X_{tt} = \left(\frac{1-x}{x}\right)^{0.9} \left(\frac{\rho_g}{\rho_l}\right)^{0.5} \left(\frac{\mu_l}{\mu_g}\right)^{0.1} \quad (25)$$

h_{eb} is the nucleate boiling contribution term and is computed with an empirical correlation to the ratio of the working pressure to the critical pressure, derived from Stephan and described in (Odeh et al., 1998). The nucleate boiling corrective term S is computed as a function of E and the liquid Reynolds number, an empirical correlation from Gunger & Winterton and described in (Odeh et al., 1998):

$$S = 1/[1 + (1.15E - 6 \times E^2 \times Re^{1.17})] \quad (26)$$

2.4.3 Closure equations: flow properties

Flow properties are computed using the IAPWS IF97 water/steam tables and functions. As pressure and enthalpy are computed for each tube segment and each time step, those state variables are used as argument to call properties functions like temperatures, densities, steam fractions, thermal capacities and conductivities, viscosities, etc.

2.5 Other pressure drops

Pressure drops outside the collectors, ie. in the connections between them, are modeled with specific

ThermoSysPro pressure drop components. The experimental pressure data include the loop inlet and outlet pressures, and the pressure drops for each collector. One can then extract the loss induced by the connections between the collectors, and use them to compute pressure drop coefficients to be used in the singular loss models. The coefficient can then be manually adjusted to match the experimental data.

2.6 Boundary conditions

The inlet of the collectors line, along with the small injection cooling at the last collector inlet, are modeled as a flow source with imposed mass flow rate values and imposed specific enthalpy values. Those values are directly extracted from the DISS loop experimental data.

The flow outlet of the collectors line is modeled as a pressure sink, with imposed values also directly extracted from the experimental data.

The parabolic collector inputs, direct normal irradiation, ambient temperature, and incidence angle are also directly extracted from the experimental data. As these data are given physical sensors, they require some smoothing with signal processing tools, for the sake of simulations stability.

3 Simulation of a clear sky day

A first simulation is carried out with the described model and the input data of a good sunny day of April. Figure 5 shows the measured DNI at the DISS test site on April 22, 2002. Data start at 09:00:00 and the collectors are defocused at 15:56:40, so the simulation is carried out on this time range.

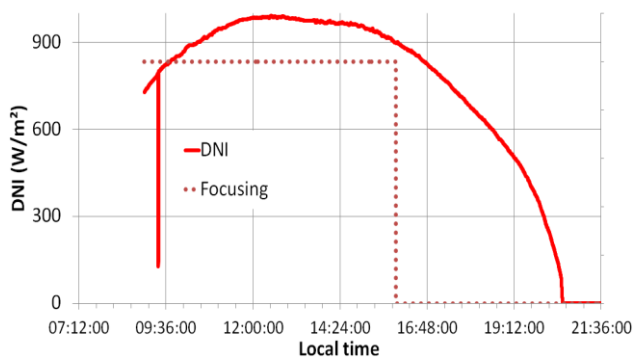


Figure 5: DNI and collector focusing of April 22, 2002

3.1 Input data for boundary conditions

3.1.1 Collector optical model input

As previously stated, the measured DNI is directly used as input data in the collector optical model. The measured ambient temperature is also used as input to the model, but for the sake of simulation stability, its noisy signal is interpolated with a 5th degree polynomial, as pictured on figure 6.

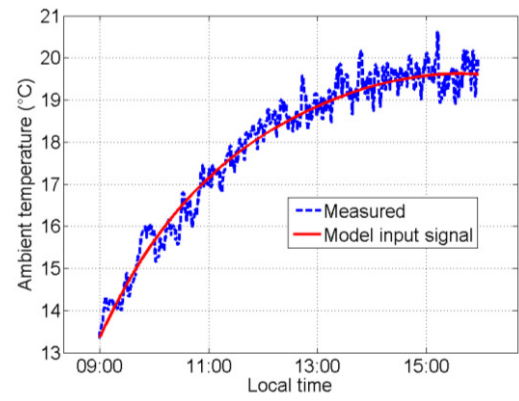


Figure 6: Ambient temperature measured data and model input

The sun incidence angle θ evolution for April 22, 2002 is extracted from the MeteoNorm database (location: Almeria airport) with an hour time step.

3.1.2 Flow inlet

The measured mass flow rate is used as input in the mass flow rate source of the model. The data is processed with a sliding averaging function to smooth the signal, as shown on figure 7. The test loop is given a temperature setpoint change during the operation, which is why two main evolution sections are visible on the inlet flow rate plot.

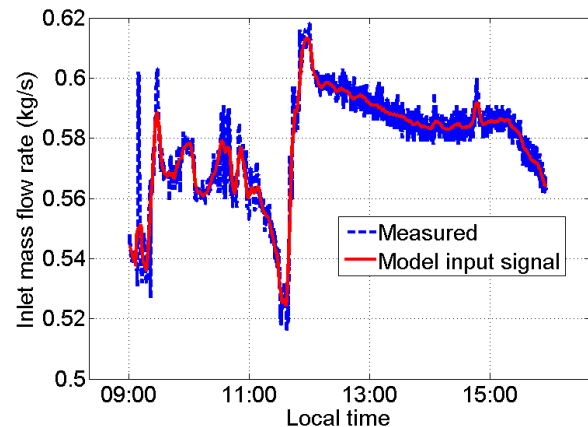


Figure 7: Inlet mass flow rate measured data and model input

For the energy state at the inlet, the ThermoSysPro flow source component requires the specific enthalpy as an input. Available experimental data including temperature and pressure at the first collector inlet, specific enthalpy is computed from those values with the IAWPS IF97 tables, and used as model input.

The injection cooling at the inlet of the last collector, whose role is to keep the outlet temperature on setpoint, is modeled the same way. Its flow rate evolution can be seen on Figure 8.

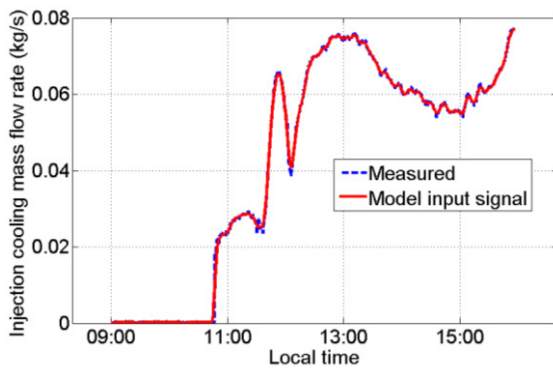


Figure 8: Injected mass flow rate for cooling

3.1.3 Flow outlet

The imposed pressure at the collector outlet also comes from experimental data. The outlet pressure control valve is closed until the loop reaches the setpoint pressure at the outlet (loop is then said to operate in sliding pressure mode), then the valve is controlled to keep the pressure at the setpoint (which is an operation in constant pressure mode). For the simulation, no control valve is modeled, and it is simply the outlet pressure that is taken as boundary condition. The pressure field in the loop is then computed from the outlet value and the pressure drops models. As can be seen on Figure 9, the DISS loop is operated to about 31 bars on that day.

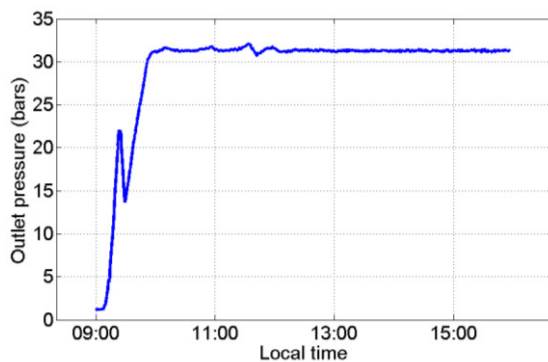


Figure 9: Loop outlet pressure evolution

3.2 Results and discussions

3.2.1 Pressure field in the loop

Figure 10 shows the pressure at the first collector inlet, thus representing the overall pressure loss in the collector.

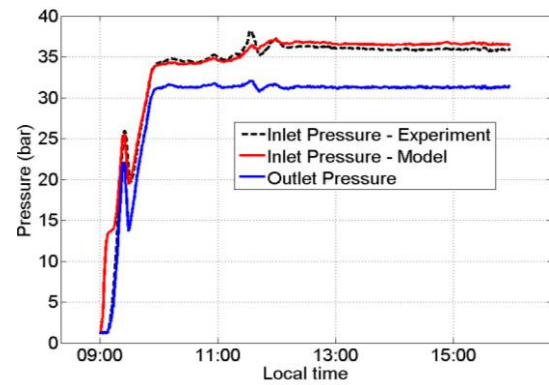


Figure 10: Inlet and outlet pressure in the loop

It can be seen that the computed pressure inlet of the model is quite close to the experimental value, with a small over-prediction of about 1 bar at nominal operation. The dynamic behavior resulting from the change in the boundary conditions is also well described, although the model pressure rises more fastly than the experiment. We assume that this difference is due to the fact that temperatures in the last collectors reach saturation level more fastly in the model (as can be seen in the next section temperature plots), since computation starts with higher enthalpy levels than the experiment (for solver stability reasons). Therefore, if vaporization starts more quickly in the model, a higher pressure drop is observed. The fact that the inertia term of the momentum balance equation (10) is set to zero can also explain this difference between model and experiment, as well as the delay of pressure drop peaks between model and experiment, visible on the following figures.

It seems also interesting to compare specific pressure drops in some collectors. Figure 11 shows the pressure drop inside collectors 1 and 3. The model clearly under-predicts the pressure loss of collector 1, where flow is only liquid, whereas the prediction is rather good for collector 3, where the flow has two phases. Figure 12 shows the same data for collector 5 and 8. For those two collectors, where a two-phase flow is present, the model under-predicts the pressure loss. Finally, figure 13 shows the pressure losses for the collectors in the superheating section, collectors 10 and 11, where superheated steam is found. The prediction for the loss in collector 10 is rather good, whereas a large difference is found with collector 11. This could be explained by the fact that the model does not describe the physics of the injection cooling very well. This phenomenon produces a pressure drop that is not taken into account in the model, which is a simple energy balance flow mixing component. For collector 1, it is assumed that the large difference between model and experiment is due to the presence of steam bubbles in the first collector of the experimental loop. Indeed, although the average temperature is below saturation, it can locally reach saturation, thus generating small vapor bubbles that will quickly condensate, but will

generate additional pressure drops. Those bubbles cannot be “seen” by the model, since it uses a homogeneous flow assumption.

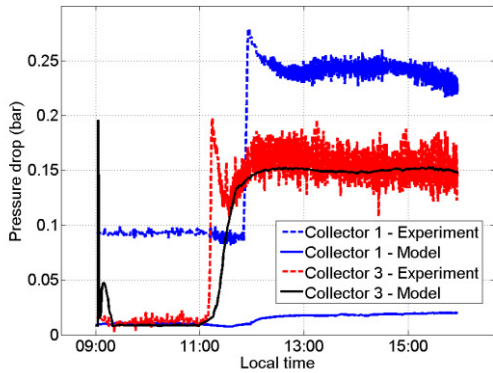


Figure 11: Pressure drop in collectors 1 and 3

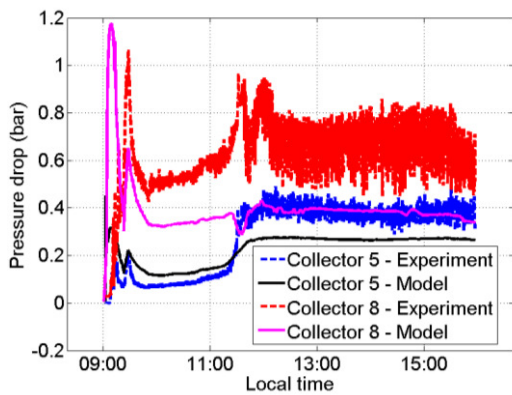


Figure 12: Pressure drop in collectors 5 and 8

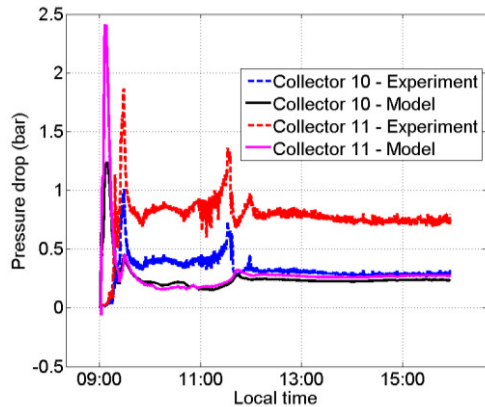


Figure 13: Pressure drop in collector 10 and 11

So it can be seen with the previous figures that although the overall pressure drop along the loop is slightly over-predicted, model pressure losses in each collector are almost always less than experimental data. It is therefore the singular pressure loss components, modelling the connections between collectors, which correct the error. In terms of dynamics, simulation results seems to show a similar behavior as measurements, but smoother.

3.2.2 Temperatures and steam fractions

Figure 14 shows the temperature evolution for some of the 8 first collectors. Collectors 2,3,5 and 8 all reach saturation temperature at about 240°C, which shows that they feature a two-phase flow. The inlet temperature of collector 1 is both the actual experimental value and the model boundary condition. For each of the other collectors, the agreement is good between experimental and modeling values.

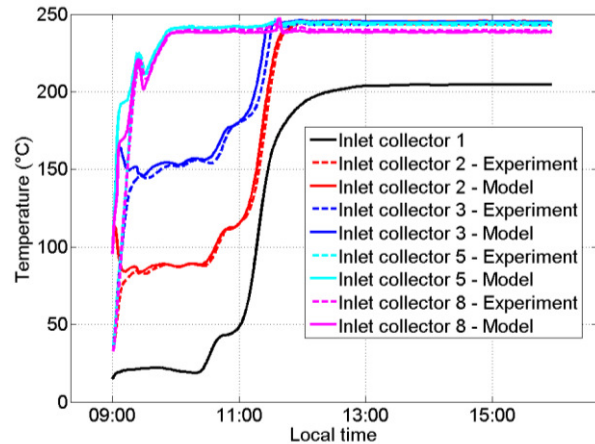


Figure 14: Inlet temperatures of collectors 1,2,3,5 and 8

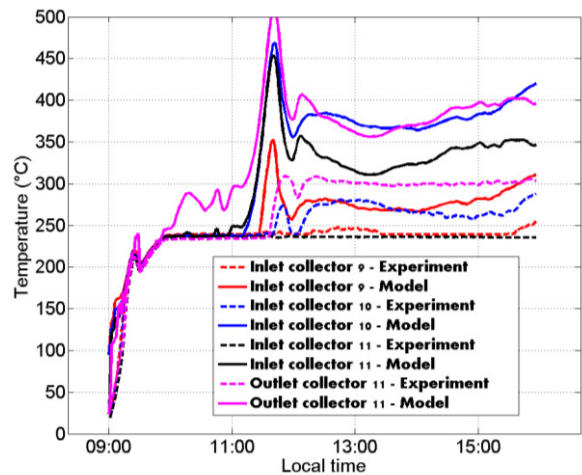


Figure 15: Temperatures of collectors 9 to 11

Figure 15 pictures the temperatures of collectors located in the superheating section of the modeled loop. The model results show indeed that at collector 9 inlet, the temperature is already greater than saturation value. It is not the case for the experimental results, which show that inlet temperature of collector 9 remains at saturation level, although briefly going over it. It means that for the model, superheating starts somewhere in collector 8. This is confirmed by figure 16 which shows the steam fraction evolution in each of the collector discrete cell. It can be seen that the fraction is 1 from cell 4 on. The experimental results show that inlet temperature of collector 10 is above saturation level, which means that superheating in the experimental loop starts somewhere in collector 9.

Therefore there is a significant difference between experiment and model of about one collector's length as to the location of the superheating "beginning". It is also and simply visible by the significant temperature difference between model and experiment for each of the collectors inlets in this superheating section. We assume that this difference comes from the fact that thermal losses are under-evaluated by the parabolic collector model, especially in the superheating section. In particular, the convection losses model uses a fixed heat transfer coefficient, when it actually is a function of many parameters and may be much higher than the value used in the model. Also, the model does not account for the thermal losses in the connections between collectors. Lower thermal losses can also explain the larger temperature peaks visible with the model results. In the experimental loop, larger thermal losses prevent large temperature peaks. Thermal losses can also be under-evaluated in the vaporizer section, but the effect is not visible since temperature remains at saturation value.

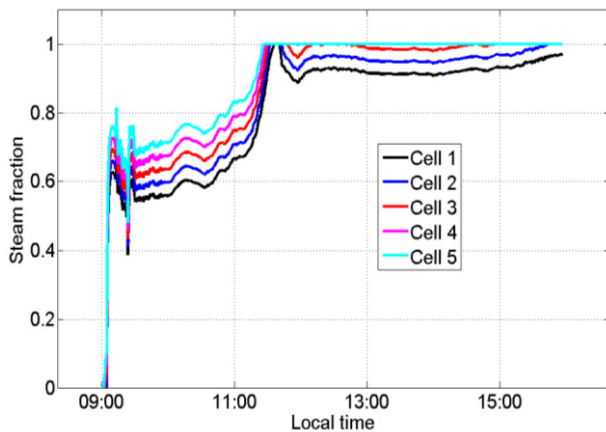


Figure 16: Steam fraction evolution in collector 8

Another source of error is probably the modeling of the injection cooling, or "desuperheating". It is modeled as a simple enthalpy balance component, whereas a complex atomization process actually takes place, with physical phenomenon that are not described by this type of modeling, and which are beyond the scope of this work.

A first general calibration of the model is done by modifying two coefficients. The first one is the modification from the original value of the convection thermal losses coefficient h . The coefficient is computed using a free convection correlation (a no-wind situation is assumed) extracted from Chan and described in (Forristall, 2003). The second modification is done on the peak optical efficiency of the collectors: it is reduced by 5%, from 73% to 68%. As can be seen on the temperature plots of Figure 17, model results are then significantly improved.

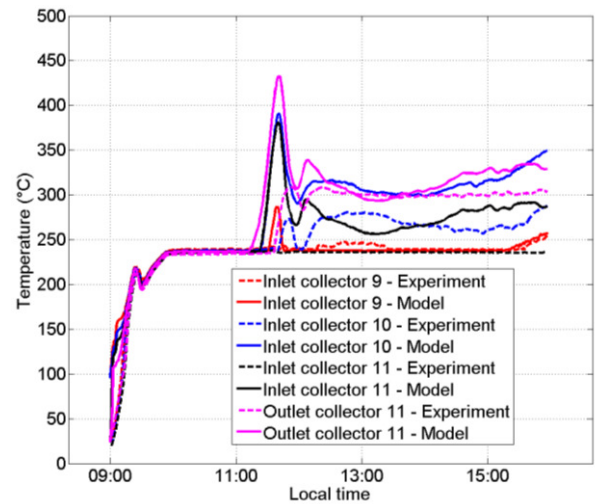


Figure 17: Temperature of collectors 9 to 11, after general model calibration

Better agreement is found for collector 9, the simulated inlet is at saturation temperature. Agreement is also slightly better for collector 10 and at collector 11 inlet, but the difference remains large, in collector 11 in particular. Also, it can be noted that simulated temperatures in collector 11 show transient behaviors that are significantly different from measurements. Since outlet steam conditions are particularly important for DSG systems, it can be stated that the accuracy of those results is not sufficient.

Those remarks, along with previous remarks made about pressure drops, highlight the fact that precise collector-wise calibration should be made in order to improve model performance (indeed, pressure drops coefficients and thermal losses coefficients are assumed to be respectively equal in every collectors, when they are most likely different), and further simulations should be done with a focus on the transient behavior.

4 Conclusions

A Modelica model of the direct steam generation parabolic-trough experimental loop DISS has been developed. Simulations have been carried out, using experimental data from the loop as model input and boundary conditions. For the simulated April sunny day, results show a good general behavior agreement between model and experiment, but adjustments have to be made for a better fit to the experimental data, since outlet steam conditions are important in DSG systems. Special attention will be given to computed flux from the optical model in the parabolic collector model, and to calibration of losses coefficients collector by collector. Also, the simulation of a cloudy day with irradiation transients remains to be done, with full dynamic modeling of the two-phase flow.

The perspective of this work and the validation of the model is the study of advanced control strategies for the handling of irradiation transients, which are key

to the use of direct steam generation in linear CSP plants. Indeed, knowledge of the dynamics taking place in DSG systems is useful for the parameterization of the control loops.

Simulations with the nuclear two-phase flow code CATHARE are also currently being carried out, for comparison with the Modelica models.

Acknowledgements

The authors would like to thank their fellow colleagues of CIEMAT for providing the experimental data of the DISS test loop. This collaboration was made possible thanks to the funding from the European Energy Research Alliance (EERA) with the European project N° 609837 “Scientific and Technological Alliance for Guaranteeing the European Excellence in Concentrating Solar Thermal Energy - STAGE STE”.

References

- Eck, M., Benz, N., Feldhoff, J. F., Gilon, Y., Hacker, Z., Müller, T., Riffelmann, K.-jürgen, et al. (2008). The potential of direct steam generation in parabolic troughs - results of the German project DIVA. *Proceedings of the 14th Biennial CSP SolarPACES Symposium*.
- Feldhoff, J. F., Eck, M., Benitez, D., & Riffelmann, K.-jürgen. (2009). Economic Potential of Solar Thermal Power Plants with Direct Steam Generation compared to HTF Plants. *Proceedings of the ES2009 Conference* (pp. 663-671).
- Forristall, R. (NREL). (2003). *Heat Transfer Analysis and Modeling of a Parabolic Trough Solar Receiver Implemented in Engineering Equation Solver Heat Transfer Analysis and Modeling of a Parabolic Trough Solar Receiver Implemented in Engineering Equation Solver*.
- OECD/IEA. (2014). *Technology Roadmap Concentrating Solar Thermal Electricity*.
- Odeh, S. D., Morrison, G. L., & Behnia, M. (1998). MODELLING OF PARABOLIC TROUGH DIRECT STEAM GENERATION SOLAR COLLECTORS. *Solar Energy*, 62(6), 395-406.
- Valenzuela, L., Zarza, E., Berenguel, M., & Camacho, E. F. (2004). Direct steam generation in solar boilers, using feedback to maintain conditions under uncontrollable solar radiations. *IEEE Control Systems Magazine*, 15-29.
- Valenzuela, L., Zarza, E., Berenguel, M., & Camacho, E. F. (2005). Control concepts for direct steam generation in parabolic troughs. *Solar Energy*, 78, 301-311. doi:10.1016/j.solener.2004.05.008

Transient Simulation of the Power Block in a Parabolic Trough Power Plant

Heiko Schenk¹ Jürgen Dersch² Tobias Hirsch³ Thomas Polklas⁴

¹German Aerospace Center (DLR), Institut of Solar Energy, Germany,

{Heiko.Schenk, Juergen.Dersch², Tobias.Hirsch³}@dlr.de

⁴MAN Diesel & Turbo SE, Process Industry/Engineering Steam Turbines, Thomas.Polklas@man.eu

Abstract

In the field of concentrated solar power (CSP) plants, parabolic trough systems with thermal oil as heat transfer fluid represent the technically and economically most mature technology. Due to storage systems these plants produce electricity on demand. However, a considerable portion of the annually collected thermal energy is consumed for the start-up procedure. In fact, after shut-down periods thermal masses must be reheated and additionally further energy losses due to imperfect start-up procedures occur. The present work has been carried out within the TURIKON project. The main goal is to evaluate and to optimize the transient behavior, namely the start-up of parabolic trough plants with thermal oil. For this purpose, a dynamic model was developed. An internal DLR solar library was used for the modelling of the solar field while the power block is modelled with the publically available ThermoPower library where some components had to be adapted for the needs of CSP plants. In the present publication first results are shown in order to demonstrate the capabilities of the plant model. The dynamic behavior of the power plant during normal operating mode and during a warm and a hot start-up procedure is evaluated and the warm start-up procedure energetically optimized.

Keywords: transient power block simulation, parabolic trough, concentrated solar power

1 Introduction

With the penetration of fluctuating renewable energy resources, such as wind and photovoltaic, dispatchability gets more into focus. Concentrated solar power plants offer the possibility to produce electricity on demand due to their cost-effective thermal storage systems. In the sector of point focus systems solar tower power plants aim at high process temperatures, using molten salt as heat transfer fluid or gas in order to operate high temperature gas turbines. In the sector of parabolic trough plants, the aim is to develop high-temperature processes with direct steam generation or molten salt as heat transfer fluid. However, parabolic trough power plants with thermal

oil represent the state-of-the-art. With more than 2 GW_{el} installed in Spain (Protermo Solar, 2015), these plants are the economically most mature technology amongst all CSP system. In these plants thermal oil is heated in the solar field and thermal power is transferred to a conventional water-steam cycle, where electricity is produced. Several plants are equipped with a two-tank molten salt storage system which is connected with the oil circuit with a heat exchanger. The storage system allows decoupling the electricity production from the solar energy input. However, the process parameters of these plants are limited due to the decomposition temperature of the thermal oil of 400 °C.

The modelling and simulation work presented in this paper has been carried out within the TURIKON project. In this project DLR, MAN Diesel & Turbo and the University of Duisburg Essen work jointly together. Among other goals, the aim is to examine and to optimize the transient operation and the start-up procedure of solar field and power block. In fact, the thermodynamic behavior of both subsystems is characterized by their thermal inertia that stems from fluid and steel masses. These thermal masses have a smoothing effect on the electricity production during cloudy periods. After plant shut-down, fluid and steel masses cool down individually. During the start-up process they have to be heated-up. The energy consumption of the start-up processes can account for 5 to 10 % of the annual heat production, depending on the plant's configuration and meteorological conditions. From here, it becomes clear, that the start-up procedure presents a considerable energetic optimization potential

In previous works on the start-up procedure of parabolic trough plants with direct steam generation, (Hirsch *et al*, 2006), and with thermal oil, (Hirsch *et al*, 2012), a DLR-internal Dymola library was used for the modelling of the solar field. The present publication focuses on the modelling of the power block, for which the freely available ThermoPower library was used with some adaptations and was coupled with the DLR-internal solar library.

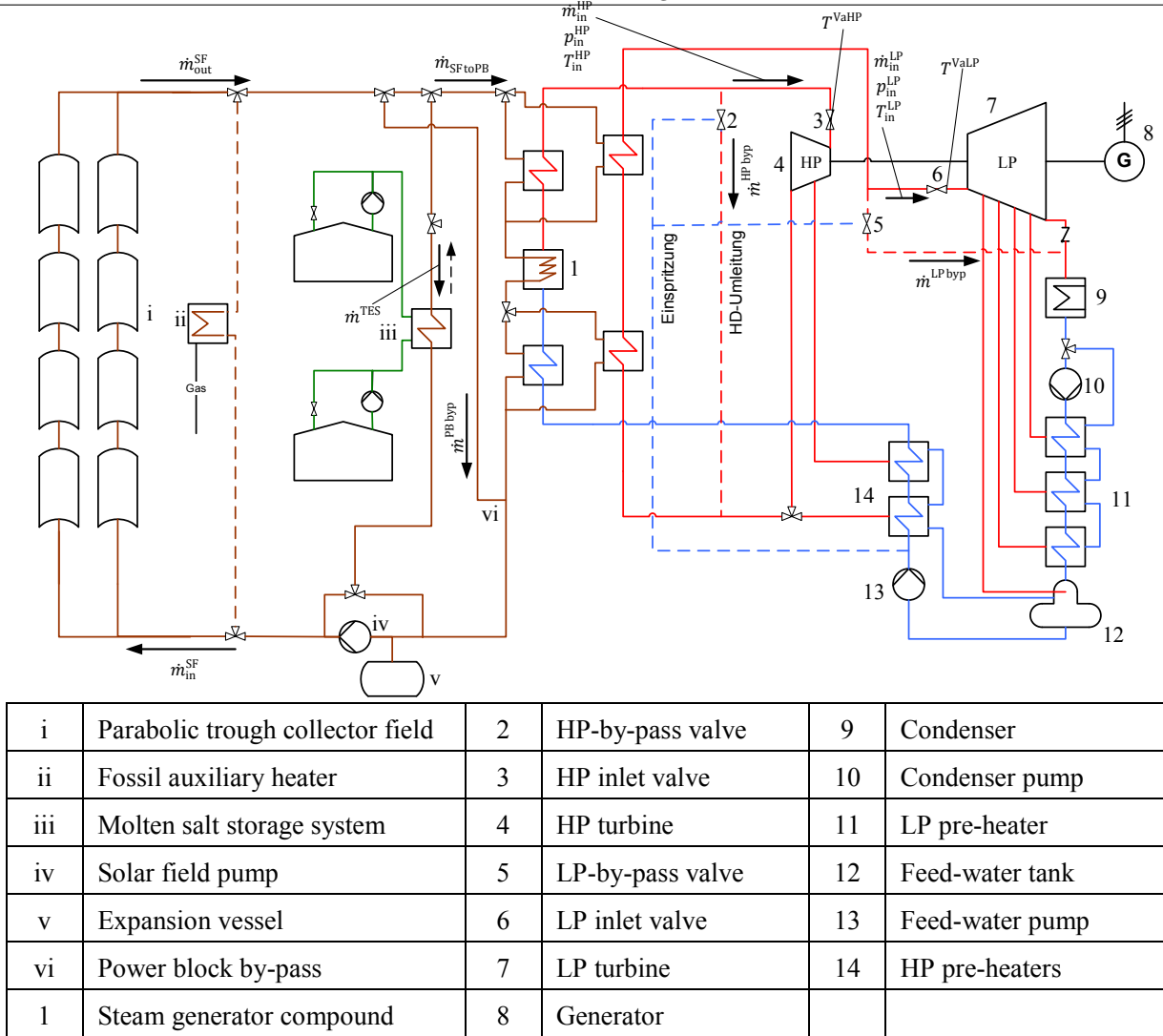


Figure 1. Hydraulic plant scheme

2 Reference System

For the modelling and simulation a parabolic trough power plant similar to the commercial systems in Spain was chosen. Figure 1 shows the hydraulic layout of the plant with all relevant components. Table 1 gives an overview of relevant thermodynamic parameters, as well as heat transfer fluid (HTF) and steel masses. In the same manner parameters of the power block are shown in Table 2. As already explained, the heat transfer fluid is heated up in the solar field and the heat is transferred through the steam generator (SG) to the power block (PB). There, electricity is produced in a conventional water-steam cycle. Heat can also be transferred to the thermal energy storage (TES) system via a heat exchanger. The sensible heat is stored in molten salt, which is pumped from the cold to the hot tank. The flow direction is reversed for discharge mode). During start-up, in case the thermal power must be limited, the power block by-pass is activated.

The steam generator consists of an economiser, evaporator, superheater and 2 reheaters, one for low and one for high temperature. There are two high-pressure (HP) turbine stages and 5 low-pressure (LP)

turbine stages. The power block is equipped with a regenerative feed-water system, where the feed-water tank is connected to the first bleed of the low pressure turbine and is situated between low-pressure pre-heaters and feed-water pump.

Table 1. Parameters of the solar circuit

Parameter	Specification
HTF	VP1, thermal oil
Collector	Eurotrough
Length of collector	150 m
Aperture Width of Collector	5,77 m
Total number of collectors	576
Nominal SF mass flow, \dot{m}^{SF}	1053 kg/s
Nominal SF temperature (inlet/outlet), $T_{in}^{SF}, T_{out}^{SF}$	293°C; 393°C
Thermal power at 837.5 W/m ² (perpendicular irradiation)	~ 285 MW
TES capacity	7 h
total steel mass	965 t
total HTF mass	1390 t

Both turbines are equipped with a by-pass valve and an inlet valve. In nominal operating mode the inlet valves are fully open and the by-pass valves are closed and the power block works in sliding pressure mode. During start-up procedure pressure and temperature gradients at the inlets of the turbines and in the steam generator are restricted. Furthermore, depending on the phase, live- and reheat steam mass flow is restricted. Live-steam and reheat mass flow are therefore controlled with the inlet valves. At the same time, live-steam and reheat steam pressure can be limited by opening the by-pass valves. In that manner steam is transferred to the condenser where it is finally condensed and heat is dissipated leading to a pressure reduction.

Table 2. Parameters of the power block

<i>Parameter</i>	<i>Specification</i>
Nominal gross electric power	50 MW
nominal thermal power	125 MW
nominal gross efficiency	39.9 %
nominal live-steam parameters $p_{in}^{HP}, T_{in}^{HP}, \dot{m}_{in}^{HP}$	101 bar; 381 °C; 53 kg/s
nominal steam parameters re-heater, p_{in}^{LP}, T_{in}^{LP}	20 bar; 381 °C; 45 kg/s
No. of turbine stages	2 HP, 5 LP
Nominal operating Mode	sliding pressure
SG volume water / steam	70 m ³
SG volume HTF	115 m ³
SG steel mass	435 t
FWS volume water / steam	80 m ³
FWS steel mass	50 t

The split design of reheaters is not common in all commercial plants but the standard configuration of the Danish company Aalborg CSP, see (Aalborg CSP, 2015). Furthermore, unlike some other parabolic trough plants, that are equipped with tube-and-shell heat exchangers, Aalborg CSP only delivers header-type heat exchangers for this application. Aalborg's evaporator is realized as two separate evaporator vessels (with thermal oil in the internal piping and water in the external pass) and a steam drum connected with so-called risers and downcomers. The evaporator works with natural circulation. Aalborg CSP provided DLR with geometric and thermodynamic data of their steam generators. The data is confidential, but can be used for the simulation within the Turikon project.

The turbine configuration is similar to a commercial system of MAN Diesel & Turbo. MAN provided DLR with parameters and data about restrictions during start-up procedure. There are 3 start-up procedures for cold, warm, and hot start-up. The start-up procedure depends on the temperature of the casings of the

turbines and their inlet valves. Amongst other constraints the following temperatures of the casing of the HP inlet valve are prescribed:

- hot start-up: $t > 240$ °C
- warm start-up: $t > 180$
- cold start-up: $t < 180$

The temperature of the casing depends on the cool-down process and on the insulation of the turbine components. In the example in section 4 after 3:45 hours of cool-down time, a hot start-up can be carried out, while in the second example a cool down time of 12:45 hours entails a warm start-up. Since cold start-up represents a rather rare event, in the present simulation study only that case was not examined.

3 Modelling

3.1 Solar Library of the DLR

The solar field model was built up with components from the DLR solar library. In the field of line focus systems the first modelling work in DLR with Dymola was carried out in the framework of a thesis, see (Hirsch, 2005). A comprehensive dynamic library of solar field components, namely for direct steam generation systems, was developed, see (Hirsch *et al*, 2005). This library was later extended in order to use single-phase fluids as thermal oil and also molten salt, see (Hirsch *et al*, 2012). Due to the utilization of standard Dymola connectors, such as Modelica 3.x fluid connectors, the library can be combined with standard Modelica libraries.

3.2 ThermoPower Library

The power block of the plant model was built up with components from the freely available ThermoPower library which has been developed in the Politecnico di Milano since 2003, see (Casella *et al*, 2003; Casella *et al*, 2006). The library contains dynamic models of all relevant components of conventional water-steam, gas turbine, and combined cycle power plants. Power plants also including control system and grid connection can be build up and simulated with the library. Nowadays, the ThermoPower library also support fluid connectors of the Modelica 3.x standard, see (Casella, 2015) and can hence be coupled with other Modelica libraries. The newest Version of the ThermoPower Library is 3.1 Beta 0, while for the here-presented simulations a preliminary version from 2013 was used.

In the standard version, the ThermoPower library does not support thermal oil as heat transfer fluid. Therefore, some additional components had to be developed for the here-presented plant model.

3.3 Plant Model

Figure 3 shows the first layer of the plant model in Dymola. The model is subdivided in submodels

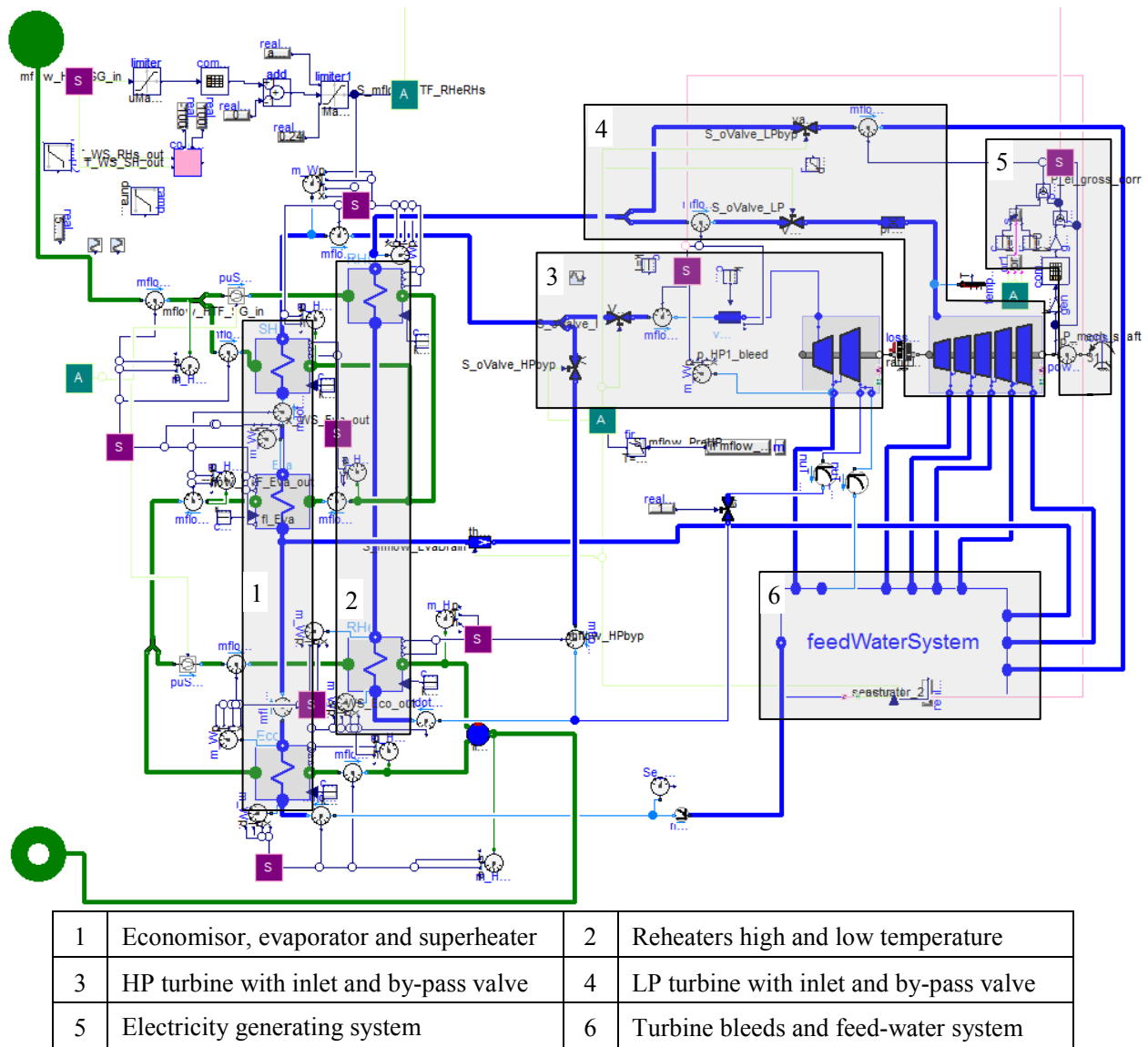


Figure 2. Model of the power block

representing the HTF circuit (solar field and storage system), the power block, and a separate control system for both. In the solar field and in the power block all major fluid and steel volumes are represented. The model comprises all components that are necessary for nominal operation and start-up procedure. In a real plant there is a multitude of additional pipes and valves, which are necessary for rare operating modes or maintenance, which are not accounted for in the model.

The pipes of the solar field are represented as one-axis discretized pipe models. As published in (Hirsch *et al*, 2010) the solar field is represented by one collector loop and a representative header system. The steel and fluid masses correspond to the ones of the real solar field. For the sake of brevity, the model of the solar field is not described here in detail.

Heat exchangers of the power block are also built up with one-axis discretized pipes models. The power block model comprises turbines, valves, pumps, heat exchangers, and vessels. The presented reference

system differs in some details from conventional water–steam power blocks. In most cases standard models from the ThermoPower library could be used with some adaptations for the modelling of the power block.

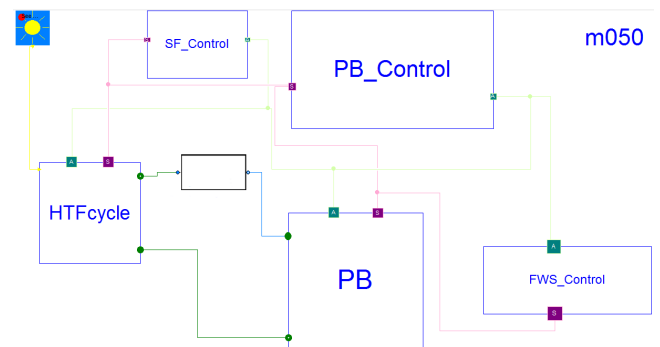


Figure 3. First layer of the plant model

In contrast to plant models in stationary simulation tools set point values cannot be prescribed. Therefore,

a comprehensive control system had to be developed. This system controls for example:

- outlet temperature of the solar field
- fill-level of vessels, e. g. in evaporator and feed-water system
- live-steam pressure and mass flow during start-up

In the following paragraphs some examples of the plants submodels are shown.

3.4 Model of the Power Block

The model of the power block is shown in Figure 2. Blue lines represent the water / steam piping and green lines the HTF piping. The two models which are used for the steam generator are described in the following sections. High and low-pressure turbines, inlet and by-pass valves are built up entirely with models from the ThermoPower library. The relation between pressure and mass flow is implemented with Stodola's law. Turbine casings and valve boxes are modelled as cylindrical steel masses.

The standard models from the ThermoPower library had to be adapted for the feed-water system. Every preheater and aftercooler, as well as feed-water tank are modelled. Only the fill-level of the pre-heaters are controlled. The bleed mass flows of the turbines adjust themselves depending on the heat exchange coefficients and the temperatures of the preheaters.

3.4.1 Economiser, Superheater, and Reheater

Figure 4 shows the model of a heat exchanger which is used for the economizer, the superheater, as well as the low and the high temperature part of the reheater.

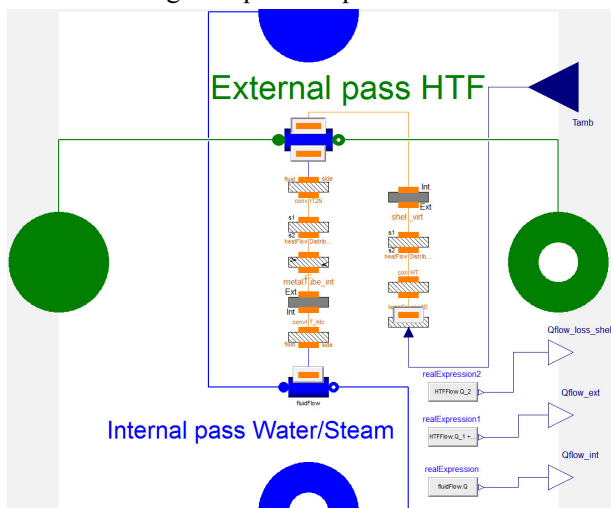


Figure 4. Model of economizer, superheater and reheater

The model is based on the heat exchanger models of the ThermoPower library, see for example (Casella *et al*, 2006). However, the flue gas on the external pass was replaced by thermal oil and the parameters had to be adapted. In the superheater and in the reheaters the internal pass is filled with steam and in the economizer

there is liquid water. External and internal pass are modelled as a representative one-axis discretized circular flow and in between there is a cylindrical steel wall. The heat transfer between the two passes is counter current. In addition to the standard model of the ThermoPower library there is also a cylindrical steel wall representing the shell of the heat exchanger. A heat loss from the shell surface to the environment is implemented, as well. The complex 3-dimensional heat transfer and the flow geometry of the tube bundle of the real heat exchanger are not represented in detail.

However, the model is using the fluid and steel volumes, the throughput-time, and the load-dependent heat transfer coefficient of the real heat exchanger and represents therefore an adequate simplification.

3.4.2 Model of the Evaporator

Figure 5 shows the model of the evaporator. Different from the economizer superheater, and reheaters water and steam is in external pass while the thermal oil circulates through the internal pass. The thermal oil flow is modelled as a circular flow. There is a cylindrical steel wall between internal and external pass representing the wall of the piping. The steam drum is modelled as a volume with steel wall that contains a water phase and a steam phase in equilibrium. As in the models of the previous section the complex 3-D geometry of the evaporator is not represented but fluid and steel volumes, and heat transfer coefficients match the ones of the real evaporator.

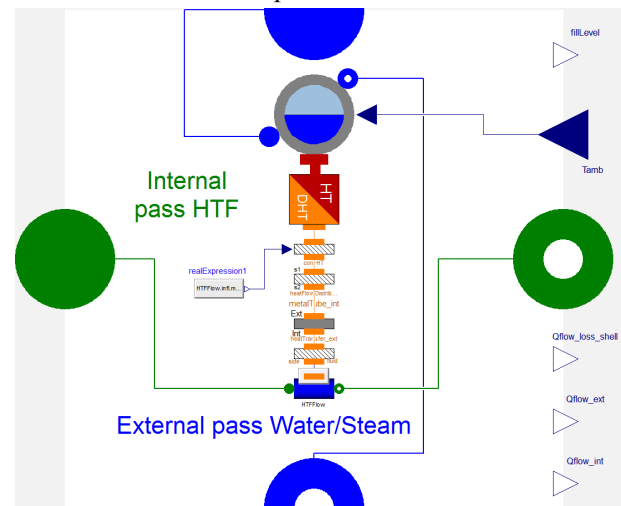


Figure 5. Model of evaporator

4 Simulation of Load Changes

A second plant model was built with the thermodynamic cycle tool Ebsilon Professional in order to derive parameters and to calibrate the Dymola model. Since Ebsilon Professional is well-respected tool in the field of steady-state analysis of thermal power plants, the here presented scenario serves as a validation for the Dymola power block model in stationary operating points.

Figure 6 shows a step-wise simulation of part-load points from 50 MW_{el} to 10 MW_{el} with the Dymola model. The HTF mass flow through steam generator and hence the thermal power to the power block was adapted in order to adapt the electrical power. The adaption time of the mass flow is set to two minutes. In the same diagram the electrical power of the steady-state model is shown. With identical gross electrical power in stationary conditions, the corresponding thermal power delivered to the power block deviates between both models by less than 0.5 % in full-load and less than 2 % in part-load. The differences between both models are due to different model assumptions namely in the part-load calculation. Since the focus of the Dymola model is the simulation of transient behavior this deviation is considered as acceptable.

Due to its inertia the Dymola model needs a settling time, t_{sett} to reach stationary conditions. As a criterion, the point in time is chosen when the electric power $P_{\text{el}}(t)$ differs by less than 1 % from the final value $P_{\text{el,final}}$:

$$\left(\frac{|P_{\text{el,final}} - P_{\text{el}}(t)|}{P_{\text{el,final}}} \right)_{(t_{\text{sett}} + t_{\text{ramp}})} < 0.01 \quad (1)$$

Since, the ramp of the load change takes 2 minutes t_{ramp} is deduced.

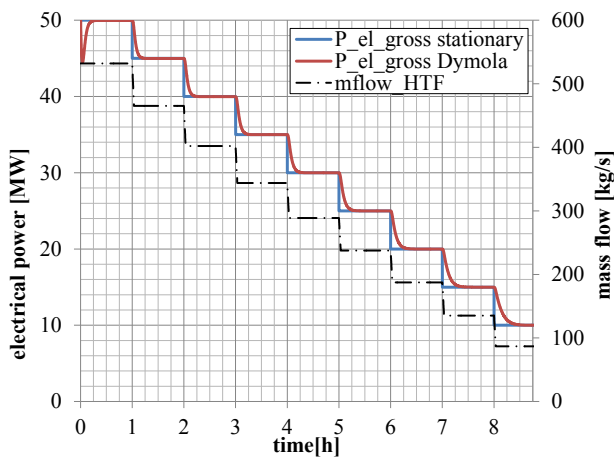


Figure 6. Electric power and mass flow

From Figure 6 it can be concluded that the settling time is load-dependent. Three examples are given:

- from 50 to 45 MW, 6 min
- from 35 to 30 MW 9 min
- from 15 to 10 MW, 25 min

The settling time is much longer, in part load operating points. The reason for that is that the heat stored or released from the fluid and steel masses is, compared to the totally transferred thermal power, much more important in part load than in full load.

5 Start-up Simulation

5.1 Heat Balance and Start-up Losses

A heat balance is established in this section in order to evaluate the energy consumption of the start-up procedure. For this purpose the thermodynamic state of the plant is introduced, see equ. (2). Q_{state} represents the total heat in all fluid and steel masses in the system compared to a reference state. For water, usually, liquid state at 0 °C is taken as reference. However, the choice of reference state is arbitrary and not of importance since in the present paper only the difference between two states is calculated.

$$Q_{\text{state}} = \sum_1^i (u_{f,i}(T_i, p_i, x_i) * m_{f,i}) + \sum_1^j (c_{s,j}(T_j - T_{\text{ref}}) * m_{s,j}) \quad (2)$$

with:

T, p, x	temperature, pressure and steam quality
T_{ref}	reference temperature
m_f, m_s	fluid and steel mass
u_f	specific internal energy of fluid
c_s	specific heat capacity of steel

After shut-down the plant cools down and this heat must be compensated during start-up. The theoretical energy for the heat-up phase, index HU, of the Solar field, index SF, of all thermal masses between cold state, at point in time t_0 , and the hot state, at t_1 , writes:

$$Q_{\text{HU}}^{\text{SF}} = Q_{\text{state}}^{\text{SF}}(t_1) - Q_{\text{state}}^{\text{SF}}(t_0) \quad (3)$$

For the power block accordingly:

$$Q_{\text{HU}}^{\text{PB}} = Q_{\text{state}}^{\text{PB}}(t_1) - Q_{\text{state}}^{\text{PB}}(t_0) \quad (4)$$

The heat-up energy also represents the minimal thermal energy need for a start-up procedure in case no other heat losses occur. However, during the start-up procedure of the solar field, heat losses of headers and piping occur, index l,HP. Furthermore, the solar field must partly be defocused, index l,def, in some cases in order to avoid overheat of collectors. Hence, the total start-up energy, of the solar field including these losses writes:

$$Q_{\text{SU}}^{\text{SF}} = Q_{\text{HU}}^{\text{SF}} + Q_{\text{l,def}} + Q_{\text{l,HP}} \quad (5)$$

During the power block start-up heat is dissipated when the turbine by-passes are open, as already described in section 2. The power block start-up energy hence is the sum of the heat-up energy and the energy that is lost due to by-pass operation, index l,byp:

$$Q_{SU}^{PB} = Q_{HU}^{PB} + Q_{l,byp} \quad (6)$$

The total energy for plant start-up then writes:

$$Q_{SU} = Q_{SU}^{PB} + Q_{SU}^{SF} \quad (7)$$

As defined in (Hirsch *et al*, 2012) the relation between the minimal start-up consumption (equals the heat-up energy) and the start-up consumption of the real process writes

$$\psi = \frac{Q_{SU}}{Q_{HU}} \quad (8)$$

ψ is always greater than 1 and can be used to parameterize simplified models for annual electrical yield analysis calculation.

5.2 Test Scenario

A scenario was chosen in which daily operation, cool-down phase, and start-up procedures can be tested. The scenario comprises two consecutive days, which are generated with a clear-sky model for a typical Spanish site for the 21st of March. The simulation results are shown in Figure 7. The upper diagram shows the direct normal irradiation (DNI) in W/m², as well as the effective DNI, which is corrected by all angle losses. Typical for a north-south aligned parabolic trough plant, the effective DNI peaks in the morning and in the evening. The diagram in the middle shows the solar field temperatures and mass flows, index SF, and the thermal oil mass flow to the steam generator, index SG. Due to model limitations the mass flow of the solar field cannot be zero. Therefore, even in the night the solar field is in recirculation mode. Since, the influence of the mass flow on heat losses is marginal this simplification does not affect the energy balance of

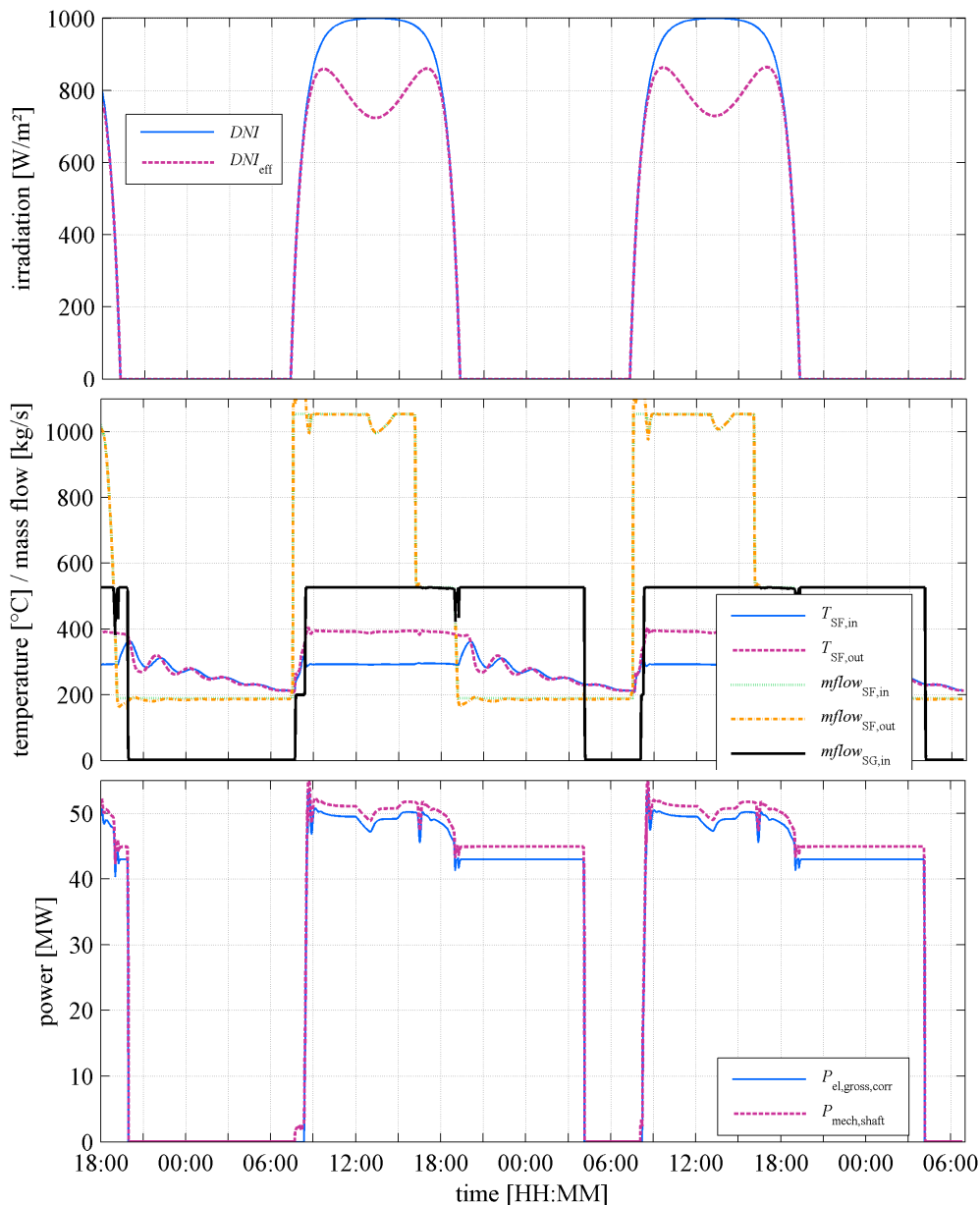


Figure 7. Plant Operation during two clear-sky days

the cool-down procedure. The lower diagram shows the mechanical shaft power of the turbine and the gross electrical power which includes the efficiency of the generator. The plant operates in a purely solar driven mode. That means that electricity shall be produced as long as possible. Load scheduling is not foreseen.

The scenario begins in the first evening with an empty storage system. The extended cool-down phase in the first night leads to a warm turbine start-up procedure during the first morning. During the first operating day, the solar field mass flow drops around 16:00 since the storage system is fully charged. In the evening after sunset, the plant is operated from storage until shortly after 4:00. During storage discharge operation, the temperature of the thermal oil is considerably lower due to the temperature difference of the heat exchanger. This leads to lower live-steam temperatures and to a reduced electrical power. The short cool-down phase of the power block entails a hot start-up procedure in the morning of the second day.

5.3 Detail Start-Up Procedure

Warm and hot start-procedures are shown in Figure 8. On the left and on the right the upper diagrams show temperatures and pressures in the water steam-cycle at the outlet of the superheater (SH), reheater superheater part (RHs), as well as the temperatures of the casing of the high-pressure turbine (VaHP) and the thermal oil at the steam generator inlet (SG).

The diagrams below show the gross electrical power relative to 50 MW in %, as well as the steam mass flows at the inlet and the by-pass valve of the low pressure turbine (LP, LPbyp) and the high pressure turbine (HP, HPbyp). As indicated as well in Figure 8, the start-up procedure is subdivided into phases A, B, C, which are between shut-down and nominal mode of the power block. For further information about the phases see Table 3. As soon as the irradiation conditions allow, the solar field is heated-up in recirculation mode. When the temperature is high enough, a part of the recirculation mass flow is deviated through the steam generator. This mass flow is called \dot{m}_{SFtoPB} (compare with Figure 1) and is set to 300 kg/s in this section. During start-up, the live- and reheat steam mass flows are controlled with the HP and LP inlet valves. The pressure is controlled, with the by-pass valves. The constraints for pressure and temperature are prescribed by the manufacturers of steam generator and turbine and are confidential. In principle in the hot and in the warm start-up procedure there are the same phases and the same constraints, however with different values.

The purpose of phase A is to preheat the casings of HP and LP turbine and to reach minimal live and reheat steam conditions. In that phase the mass flow is very small. In phase B the turbine is turned and synchronized, also with a small mass flow. The

purpose of phase C is to increase the electrical power by opening turbine inlet valves and closing the by-pass valves. During that phase gradients for mass flow, temperature, pressure, and electrical power must kept below a certain value. At the end of the phase, the power block is in nominal mode with sliding pressure.

Table 3. Phases of the start-up procedure

A	pre-heat	Task: preheat of turbine (valve) casing Limits: steam pressure and mass flows
B	Synchro- nization	Task: start-up and synchro of turbines Limits: steam pressure and mass flows
C	charge increase	Task: increase of electr. production Limits: Gradients of electrical power, pressure and temperature

The zero point of the x-axes in Figure 8 is defined as the beginning of the synchronization phase (B). During warm start-up procedure, left diagrams, phase B begins at 8:15 in morning of the first day. The preheat phase begins at 7:42 (not visible on the diagram) and phase C ends at 8:37. In the hot start-up procedure Phase B begins at 8:06 (zero point on of the x-axis). The preceding phase A takes only 1 minute since the valve boxes are already at a high temperature and the minimal live- and reheat steam conditions are attained quickly. The start-up procedure is terminated at 8:31.

The upper diagrams (Figure 8) show the ramp up of superheater and reheater temperatures and the live-steam pressure. It becomes visible, that the live- and reheat steam pressure are limited during phases A and B and are increased in phase C during the transition to the nominal operating mode. The lower diagrams (Figure 8) show the limitation of the steam mass flows during phase A and B, as well as the ramp-up in phase C. The ramp-up of the electrical power is in phase with the steam mass flows. Furthermore, the diagrams also show the by-pass mass flows which are due to the pressure control of the bypass valves. It becomes visible, that during the warm start-up procedure, more steam is condensed than in the hot start-up. The bypass steam is finally condensed. Therefore, thermal energy is dissipated which represents an energy loss.

The balance equations of section 5.1 are used in the following to establish an energetic evaluation of the start-up procedure, see Table 4. The total heat-up energy of power block and solar field, Q_{HU} sums up to 133.2 MWh_{th} and the total start-up energy, Q_{SU} to 180.7 MWh_{th} during the warm start-up. This represents almost 1.5 power block full load hours (equivalent to 125 MWh_{th}). The difference between heat-up and start-up energy, $Q_{I,SU} = 47.5$ MWh_{th} is 38 % of the energy demand for a full-load-hour and also represents the maximal optimization potential for the start-up procedure. During the hot start-up the solar field heat-up energy almost equals to the one of the warm start-

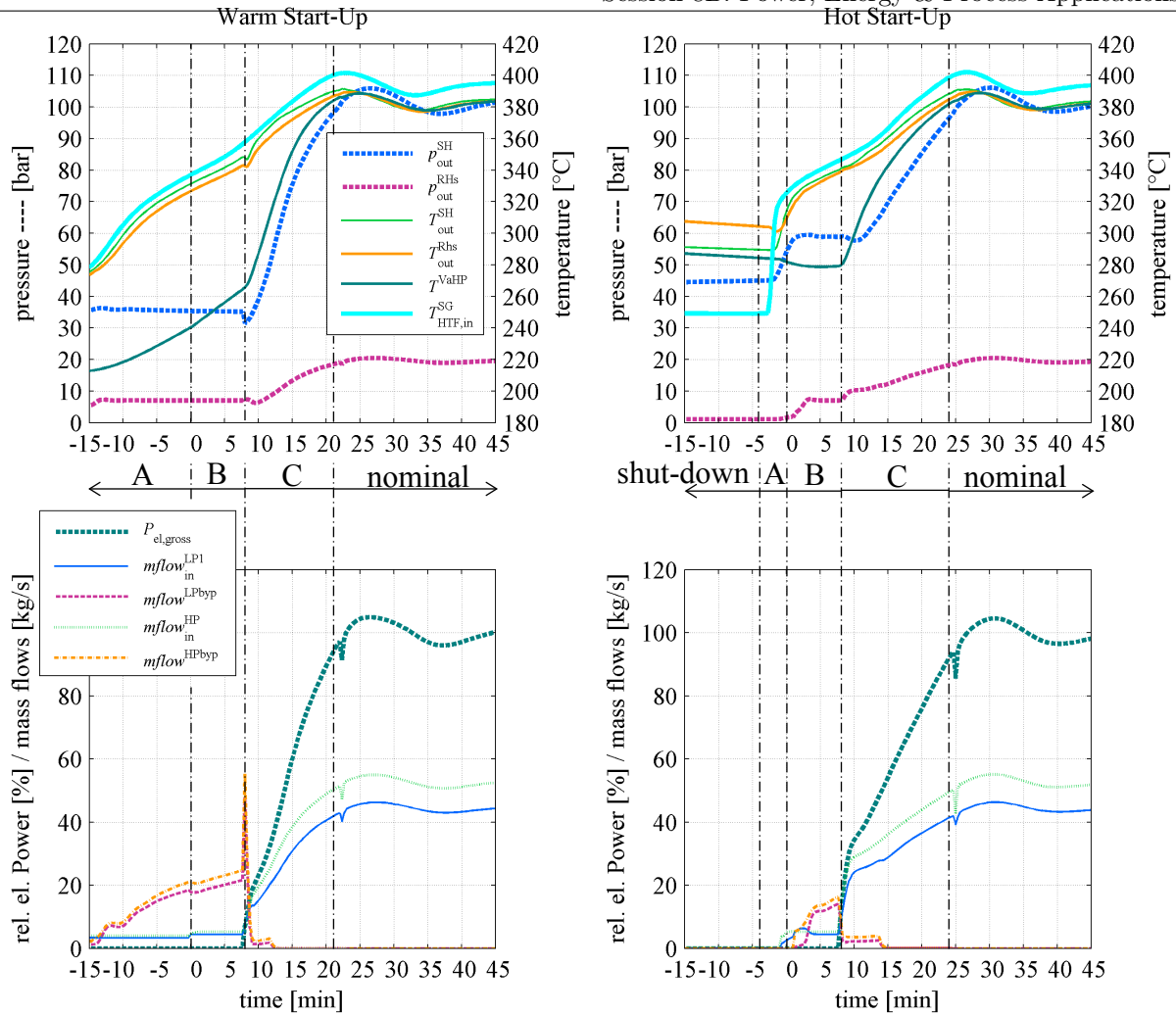


Figure 8. Warm and hot start-up. Phases: A preheat, B Synchronization, C load increase

up procedure, since the cool-down period of the solar field is similar. However, the power block start-up demands less energy due to the shorter stand-still time. In total Q_{HU} is 126.6 MWh and Q_{SU} 154.7 MWh. Therefore, the optimization potential, $Q_{I,SU}$ is 28.1 MWh representing 23 % of the energy demand for a full-load-hour. The start-up factor for the warm start-up is 1.35 and 1.22 for the hot start-up

Table 4. Energy Balance of the start-up

	Warm start-up	Hot start-up
$t_{\text{stand-still}}$ [h]	11:45 h	3:55 h
Q_{HU}^{SF} [MWh]	114.2	114,0
Q_{HU}^{PB} [MWh]	19.0	12.6
Q_{SU}^{SF} [MWh]	135.8	134.0
Q_{SU}^{PB} [MWh]	44.9	20.7
$Q_{I,SU}$ [MWh]	47.4	28.1
ψ [-]	$\frac{180.7}{133.2} = 1.35$	$\frac{154.7}{126.6} = 1.22$

5.4 Optimization of the start-up procedure

The previous section has shown possible reduction potential of energy losses during start-up. On the one

hand, the surplus power that is delivered from solar field to power block is dissipated due to by-passing. On the other hand, when the solar field reaches nominal operating state before the power block, losses occur due to defocusing, since also the charge power of the storage system is also limited. An minimum is supposed to be reached when power block and solar field reach nominal operating state at the same time. The thermal oil mass flow through the steam generator \dot{m}_{SFtoPB} , which controls the thermal power transferred to the power block was set to 300 kg/s in the scenario of the previous section. This mass flow is now varied in order to find energetic optimum. Figure 9 shows the evolution of the losses and their sum $Q_{I,SU}$ for \dot{m}_{SFtoPB} from 50 kg/s to 350 kg/s. As expected, with an increasing \dot{m}_{SFtoPB} more heat is delivered to the power block and more heat must be dissipated and therefore $Q_{I,byp}$ increases. At the same time more heat is drawn from the solar field, and the need to defocus, hence $Q_{I,def}$, decreases. The heat losses of the solar field piping and equipment, $Q_{I,def}$, remain almost unaffected, since they are only temperature-dependent. The minimal value of 36.7 MWh_{th} for $Q_{I,SU}$ is attained with $\dot{m}_{SFtoPB} = 100$ kg/s – compared to 47.5 MWh_{th} at 300 kg/s. The same variation was carried out for the hot

start-up procedure, which leads to a very flat optimum. Since the optimization potential of $Q_{1,SU}$ is almost negligible, the results are not shown here.

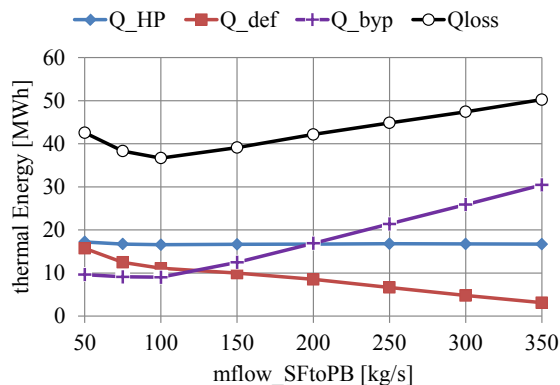


Figure 9. Energetic Start-up consumption over mass flow during warm start-up procedure

6 Conclusions and Outlook

A model of a parabolic trough power plant with thermal oil was developed with the DLR solar library and the publically available ThermoPower Library. The main focus of the modelling was put on the transient behavior, namely the start-up procedure, of the power block. With the model, the interaction between solar field, storage system and power block during start-up and the impact of transient effects is examined.

In a first step, the dynamic behavior of the power block during nominal operation with load changes is examined. As expected the thermal masses lead to a delayed behavior of the electricity production.

In a second step a scenario of two consecutive clear-sky days was chosen for a simulation. During the first night an empty storage system entails a cool-down time of almost 12 hours which is followed by a warm start-up procedure in the next morning. The totally consumed energy of the start-up procedure is 180.7 MWh_{th}. 133.2 MWh_{th} are necessary for the heat-up of the thermal masses and 47.5 MWh_{th} represent additional thermal mainly due to defocusing and bypassing. In the second night, a fully charged storage system leads to a hot start-up after a cool-down time of less than 4 hours. The energy consumption is 154.7 MWh_{th}. In both cases the heat demand for the thermal masses of the solar field is the same, since the solar field cool-down time is equal in both cases.

In a last step, the thermal oil mass flow to the power block and hence the thermal power input during the start-up procedure was varied, leading to a reduction of around 11 MWh_{th} for the warm start-up procedure.

Finally, the paper shows the potential of the dynamic plant model in Dymola. With further simulations the impact of the inertial behavior of the power block could be evaluated with regards to dispatchability. Additional start-up simulations with

modified plant configurations could provide more data on start-up consumption, for example as a function of plant size, cool-down time and irradiation conditions. Furthermore, new control strategies during start-up can be tested.

7 Acknowledgements

The authors would like to thank the German Federal State of North Rhine-Westphalia and the European Regional Development Fund for the financial support of the project TURIKON in the frame of the program progress NRW and the goal 2-program 2007-2013, Phase VI (Grant No. 64.65.69-EN-2019).

The authors would also like to thank Aalborg CSP for providing comprehensive data of their steam generators and Politecnico di Milano for making available the ThermoPower library.

References

- Aalborg CSP. Aalborg CSP steam plant configuration, 2015/04/16, www.aalborgcsp.com.
- Francesco Casella and Francesco Schiavo. Modelling and Simulation of Heat Exchangers in Modelica with Finite Element Methods, *3rd International Modelica Conference*, pp. 343-352, 2003.
- Francesco Casella and Francesco Pretolani. Fast Start-up of a Combined-Cycle Power Plant - a Simulation Study with Modelica.pdf, *5th International Modelica Conference*, pp. 7, 2006.
- Francesco Casella. ThermoPower - Open library for thermal power plant simulation, 2015/04/17, <http://thermopower.sourceforge.net/>.
- Tobias Hirsch. Fortschritt-Berichte Energietechnik, Reihe 6, No. 535, VDI Verlag, Dynamische Systemsimulation und Auslegung des Abscheidesystems für die solare Direktverdampfung in Parabolrinnenkollektoren, 2005, ISBN: 3-18-353506-8.
- Tobias Hirsch, Markus Eck, Wolf-Dieter Steinmann. Simulation of transient two-phase flow in parabolic trough collectors using Modelica, *4th International Modelica Conference*, Vol. 1, pp. 403-412, 2005.
- Tobias Hirsch and Markus Eck. Simulation of the Start-Up Procedure of a Parabolic Trough Collector Field with Direct Solar Steam Generation, *5th International Modelica Conference*, pp. 135-143, 2006.
- Tobias Hirsch, Heiko Schenk, Norbert Schmidt, Richard Meyer. Dynamics of Oil-based parabolic Trough Plants - Impact of transient Behaviour on Energy Yields, *16th SolarPACES Conference*, pp. 8, 2010.
- Tobias Hirsch, Jan Fabian Feldhoff, Heiko Schenk. Start-Up Modeling for Annual CSP Yield Calculations, *Journal of Solar Energy Engineering*, 134 (3), pp. 031004-1..9, 2012, DOI: 10.1115/1.4006268.
- Protermo Solar. Protermo Solar - Asociación Española de la Industria Solar Termoeléctrica, 2015/04/16, <http://www.protermosolar.com>.

Fault Detection and Diagnosis with Modelica Language using Deep Belief Network

Dongkyu Lee¹ Byoungdoo Lee¹ Jin Woo Shin²

¹Green City R&D Team, R&D Division, Hyundai Engineering and Construction Company, South Korea

²Department of Electrical Engineering, KAIST, South Korea

Abstract

The air handling unit (AHU) is the main component of heating, ventilation and air-conditioning (HVAC) systems, and irregular faults in AHUs are major sources of energy consumption. For energy efficient operation of HVAC, this paper aims to detect and diagnose three abnormal states in the AHU with the popular deep learning model, called Deep Belief Network (DBN), where we train it using various data generated by Modelica.

Key words: Fault detection and diagnosis, Air-handling unit, Deep Belief Network, Modelica

1 Introduction

There has been a consistent significant increase in the awareness of the importance of control strategies for heating, ventilation, and air conditioning (HVAC) systems in the building energy sectors. It is available for use energy more efficient with great qualities of monitored data and well-operated control components which are essential for achievements of entire HVAC control systems. Despite its benefits, however, energy wastes are still considered the main disadvantage with HVAC systems, and therefore, the development of fault detection and diagnosis (FDD) strategies for energy saving in buildings are considered crucial. With this, there have been many studies about FDD in HVAC systems: Massieh Najafi presented modeling and measurement constraints in fault diagnostics for HVAC systems (Najafi et al, 2012); Zhimin Du developed a wavelet neural network-based fault diagnosis in an AHU. The AHU, as one of the

main components of HVAC systems, is the heat exchange station between air and water (Du et al, 2008). With this, Modelica is used in the AHU for FDD to identify and prevent application faults and the difficulties they cause.

Generally, the methods of FDD are divided into three different categories which are rules-based, model-based and data-driven methods. The rules-based FDD methods are achieving with expert knowledge and experience rules without any mathematical models (House et al, 2001, Schein et al, 2006). Analysis of detection and diagnosis with checking rules of expert knowledge and experience is accomplished. Contrasting from rules-based methods, model-based FDD methods are attaining based on systematic physical and mathematical models (Salsbury et al, 2001, Yu et al 2002). This method is achieving detection and diagnosis of abnormal states with comparing real values with data gained from models. Nowadays, data-driven FDD method is adopted to apply due to a lot of data are available for gaining from building energy management system (BEMS). Data-driven FDD method is using historical data to detect and diagnose and include different analysis such as neural network (Wang et al, 2002, Lee et al, 2004), wavelet analysis (Du et al, 2008), and the statistic methods (Du et al, 2007, Xiao et al 2009) etc.

The aim of this research is to use a deep belief network (DBN) which is one of data-driven FDD methods achieving detection and diagnosis

Nomenclature

HVAC	heating, ventilation and air conditioning
AHU	air handling unit
FDD	fault detection and diagnosis
DBN	deep belief network
SAT	supply air temperature
RAT	return air temperature
EAT	exhausted air temperature
HIAT	heat exchanger input air temperature
HOAT	heat exchanger output air temperature
HIWT	heat exchanger input water temperature
HOWT	heat exchanger output water temperature
SAP	supply air fan power
RAP	return air fan power
SAE	supply air enthalpy
RAE	return air enthalpy
OAD	outdoor air damper
EAD	exhausted air damper
RAD	return air damper
OADT	outdoor air dry-bulb temperature
OAWT	outdoor air wet-bulb temperature
IT	indoor temperature
EAF	exhausted air flow rate
OAF	outdoor air flow rate
RAF	return air flow rate
SAF	supply air flow rate

abnormal states in AHU. If using data-driven FDD, it is important to use historical data appropriately. Therefore, data mining and machine learning are proper method to conduct FDD with historical data. The normal and abnormal data are gathered from model using Modelica. In this research, proper location and number of sensors are important to conduct FDD system AHU. It is difficult to apply faults in real system, through this procedure; therefore, Modelica is made use of applications of FDD in AHU like as a real system. Considering the data gained from model using Modelica, the machine learning framework uses two kinds of data: training data and test data. After machine learning, the DBN is used to detect and diagnosis specific faults, which will be mentioned in Section 2 of this paper. Figure 1 illustrates the detailed process of the entire fault detection and diagnosis procedure.

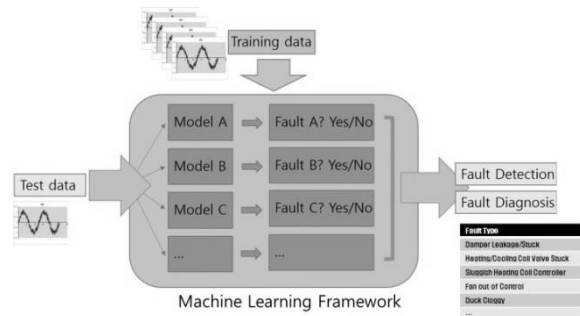


Figure 1. Process of fault detection and diagnosis using machine learning

The paper is organized in four different sections: Section 1 provides an introduction of fault detection and diagnosis with Modelica; Section 2 defines HVAC systems and discusses the faults in the system identified through Modelica; Section 3 is a method of fault detection and diagnosis using machine learning; and finally, Section 4 ends this research with a conclusion.

2 System Description

2.1 Typical system of AHU and HVAC

Figure 2 shows a typical HVAC system in a building. Here, the supply air, the mixture of the outdoor air and recycled air, exchanges heat and humidity with the chilled water in the AHU. The chilled water coming from the chillers is delivered by the pumps to the AHU. After being cooled down by the chilled water, the supply air is delivered to each air conditioning zone by the variable-speed supply fan. Moreover, the return air is divided into two streams by the variable-speed return fan: one stream is exhaust air to the outside of the building, and the other is recycled in the next air circulation (Du et al, 2014).

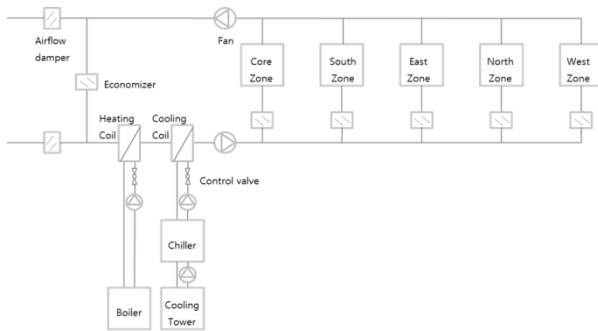


Figure 2. Typical Heating, Ventilation and Air-conditioning (HVAC) system in a building

The supply fan speed is regulated based on the duct static pressure. The return fan controller tracks the supply fan air flow rate reduced by a fixed offset. The duct static pressure is adjusted so that at least one VAV damper is 90% open. The economizer dampers are modulated to track the set point for the mixed air dry bulb temperature. Priority is given to maintain a minimum outside air volume flow rate. In each zone, the VAV damper is adjusted to meet the room temperature set point for cooling, or fully opened during heating. The room temperature set point for heating is tracked by varying the water flow rate through the reheat coil. There is also a finite state machine that transitions the mode of operation of the HVAC system among the modes: occupied, unoccupied off, unoccupied night set back, unoccupied warm-up, and unoccupied pre-cool. In the VAV model, all air flows are computed based on the duct static pressure distribution and the performance curves of the fans. Local loop control is implemented using proportional and proportional-integral controllers, while the supervisory control is implemented using a finite state machine.

To model the heat transfer through the building envelope, a model of five interconnected rooms is used. The five room model is representative of one floor of the new construction medium office building in Seoul, Korea. There are four perimeter zones and one core zone. The thermal room model computes transient heat conduction through walls, floors, and ceilings, and long-wave radiative heat exchange between surfaces. The convective heat transfer coefficient is computed based on the temperature difference between the surface and the

room air. There is also a layer-by-layer short-wave radiation, long-wave radiation, convection and conduction heat transfer model for the windows.

Each thermal zone can have air flow from the HVAC system, through leakages of the building envelope (except for the core zone) and through bidirectional air exchange through open doors that connect adjacent zones. The bidirectional air exchange is modeled based on the differences in static pressure between adjacent rooms at a reference height plus the difference in static pressure across the door height as a function of the difference in air density. There is also wind pressure acting on each facade. The wind pressure is a function of the wind speed and wind direction. Therefore, infiltration is a function of the flow imbalance of the HVAC system and of the wind conditions (ASHRAE, 2006; Deru et al, 2009; Modelica Buildings Library ; TARCOG, 2006).

2.2 Modeling of HVAC System with Modelica

Most researches accomplished FDD through simple amounts of sensors or regardless of real control logic of HVAC system. However, Modelica can make the AHU and HVAC system like as a real system. Modelica Buildings library carried out the modeling for the HVAC system of a building as shown in Figure 2 in its illustration of specific components of the HVAC system like in Figure 3.

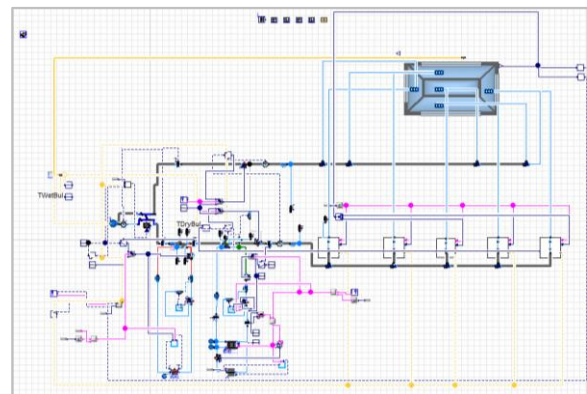


Figure 3. HVAC system accomplished by Modelica

Aside from the modeling of the entire system,

the operation logic is also necessary to operate using Modelica. As described in Section 1, this research aims to apply cooling operation only. The operation logic of cooling is as follows.

► Cooling Logic

- Cooling Coil Control Valve: proportional integral (PI) control is applied to maintain the temperature of supply air at 16°C (k: 0.01, Ti: 600 sec).
- Chiller: set temperature of chilled water at 8°C.
- Chiller on/off:
 - On: schedule of occupants, temperature range of cooling coil inlet at more than 12°C
 - Off: temperature range of cooling coil inlet at less than 8°C
- Cooling Circulation Pump: PI control to remain pressure of cooling pipe (k: 0.0005, Ti: 100 sec, flow rate: 10 kg/s)
- Fan for Cooling mode: Unoccupied night set back, unoccupied pre-cool, safety mode
- Fan for Cooling on/off: VAV and PI control to maintain pressure of indoor area at 410 Pa (k: 0.5, Ti : 15sec)

► Terminal Box Logic

Terminal Unit Dampers: PI control of damper proportion according to the set temperature of the indoor area (k: 0.1 Ti: 120 sec)

Referenced by Buildings of Library developed by LBNL 1.5/Examples/VAVReheat/Controls/Economizer.mo

2.3 Modelica Testing

This research determined specific data of sensors compared to typical available sensors to check the accuracy of fault detection and diagnosis. Table 1 described the sensors' 21 kinds of data achieved in the research. Determined data of sensors are applied through Modelica.

Data from Sensors	SAT	OAD
	RAT	EAD
	EAT	RAD
	HIAT	OADT
	HOAT	OAWT
	HIWT	IT
	HOWT	EAF
	SAP	OAF
	RAP	RAF
	SAE	SAF
	RAE	

Table 1. Sensors generated by Modelica

As shown in Figure 4, various but necessary sensors in HVAC with Modelica.

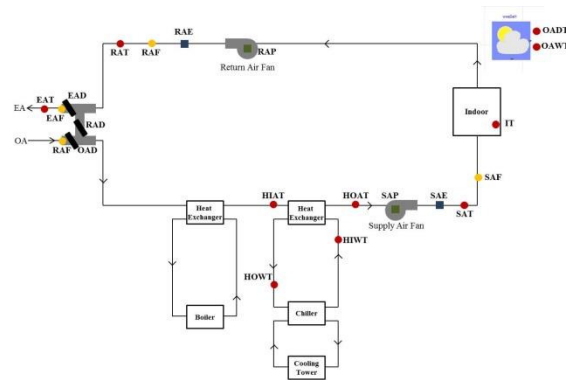


Figure 4. Location of each Sensor in Modelica

2.4 Fault Characteristics

The FDD system is applied to modern engineering fields to detect and diagnose abnormal conditions, faults, or malfunctions occurring in the routine operations of a system before these situations worsen or lead to additional damage to the entire AHU system. In the classification of faults, those with sensors and controllers are considered as one type only because feedback controllers are normally applied to modern engineering systems that mainly guarantee stability if the controller gains are suitably selected (Yuebin et al, 2014). This research focused on the three common faults of supply fans, valves, and heat exchangers. These faults are related to fans getting stuck, leakage of the cooling coil and the low efficiency of the coefficient of performance (COP) of each system. Modelica was used to change the parameters that are commonly used in normal systems.

2.4.1 Instances of Supply Fan Getting Stuck

The instance of a fan getting stuck is one of the major problems in the use of supply fans. When a fan gets stuck, flow rates through the fan are decreased. Based on this theory, the instance of a fan getting stuck is achieved by Modelica by decreasing the flow rates at 60% compared to the normal operation of a supply fan. 60% of flow rates in the fan are assumed that flow rates are decreased when the fan getting stuck. Figure 5 described the control of the flow rates of a supply fan.

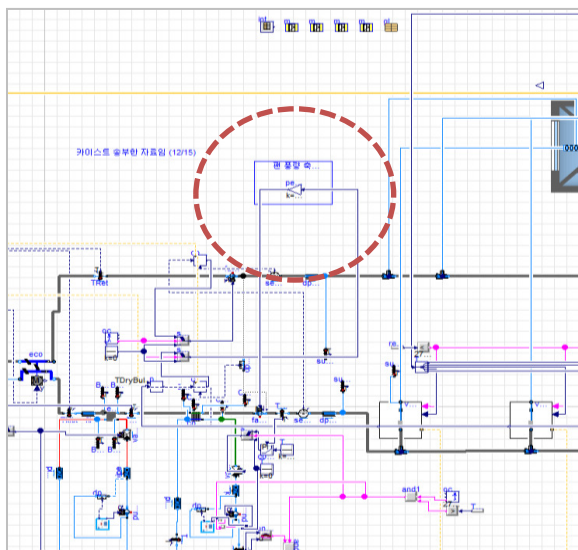


Figure 5. Implementation of supply fan getting stuck with Modelica

2.4.2 Leakage in Cooling Coil Valves

Leakage in cooling coil valves gives rise to an abnormal operation state. Modelica language can set the fault of leakage in cooling coil valves. The parameter of leakage value is represented by “L” and to accomplish the change of valve leakage from 0.0001 to 0.1 ($L=Kv(y=0)/Kv(y=1)$). “y=0” means fully closed state of the valve, “y=1” means fully opened state of the valve. Figure 6 describes the possible method to control the leakage in a cooling coil valve.

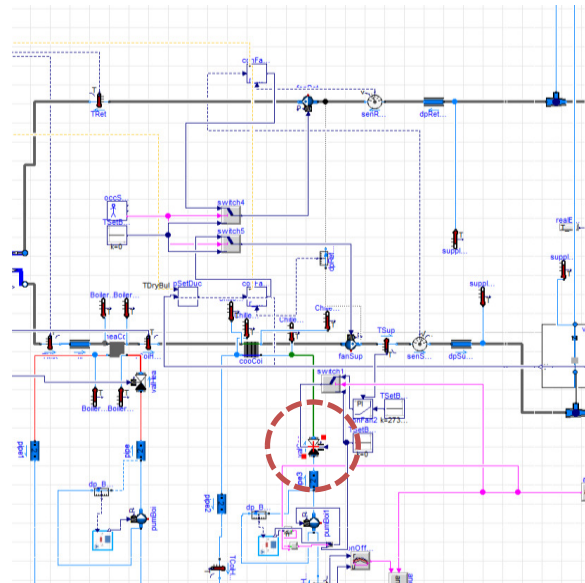


Figure 6. Implementation of leakage in cooling coil with Modelica

2.4.3 Low Efficiency of Heat Exchanger

The heat exchanger of the AHU is located between the return fan and supply fan. When the capacity of the heat exchanger is lower than that in its normal operation, the HVAC system will malfunction. Low thermal conductance means that there is low efficiency between the components of the heat exchanger. In this logic, the fault of the heat exchanger is achieved by Modelica by changing the thermal conductance from 30 kW to 15 kW. Figure 7 describes how the thermal conductance of the heat exchanger can be changed.

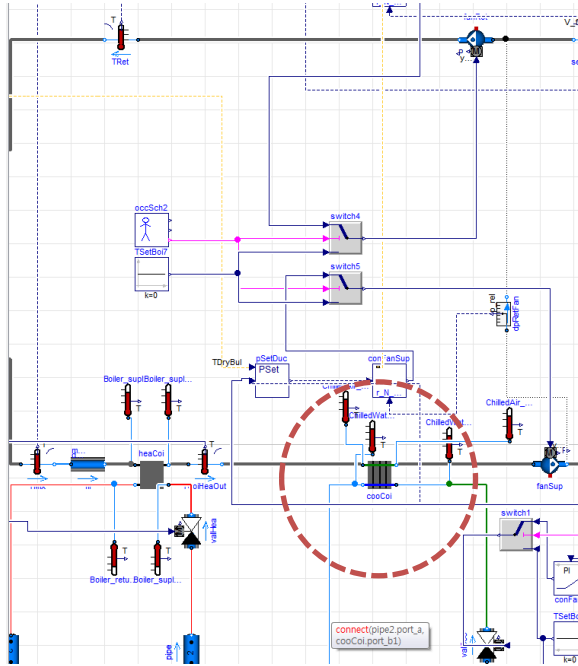


Figure 7. Implementation of low efficiency of heat exchanger with Modelica

2.5 Simulation

A total 21 kinds of data from sensors and their respective simulation with Modelica of normal state and 3 different faults states are achieved in 10 days of the summer period. Normal states of simulation are calibrated based on the logic of operation. With the results of simulation with Modelica, this research achieved various results between the normal state and the three different faults of the HVAC system. Difference between normal state and three different faults are shown from Figure 8 to Figure 10.

However, it is difficult to detect and diagnose various faults not just in the application of results of simulation in various circumstances. This means that one fault of the HVAC system has different effects on the data of sensors compared to other faults. Therefore, with machine learning, Section 3 explains the method on how to deal with wide usage involving various fault detection and diagnosis instances.

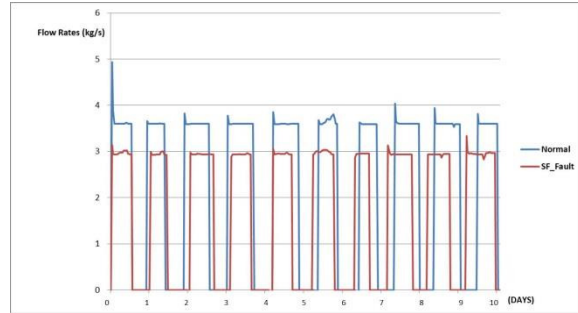


Figure 8. Comparison flow rates of return air between normal and instances of supply fan getting stuck with Modelica simulation

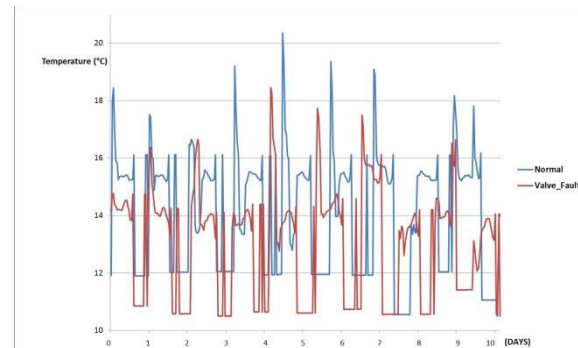


Figure 9. Comparison water temperature of heat exchanger outlet between normal and leakage of valve with Modelica simulation

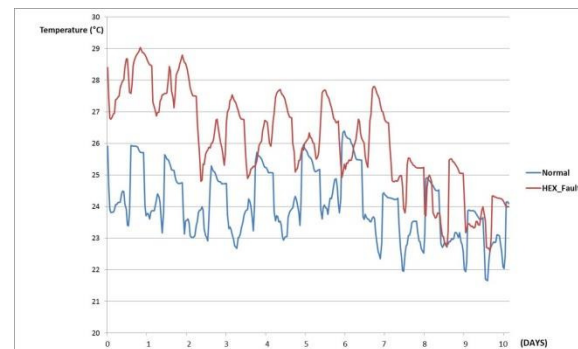


Figure 10. Comparison room temperature between normal and heat exchanger with Modelica simulation

3 Fault Detection and Diagnosis

After the application of results with Modelica as the normal and abnormal data, the fault detection and diagnosis process using machine learning is achieved as shown below. Data are filtered through the pre-process procedure, machine learning with a classifier procedure, and fault detection and diagnosis accomplished by the post-process procedure.



Figure 11. Process of FDD

3.1 Pre-processing

“Pre-processing” is a necessary procedure to minimize several irregular a number of results from sensors of AHU in Modelica due to status of building and environments. First, normalization process is essential process for the analysis of data gained by Modelica to standardize irregular data which are regardless of times and seasons into regular data. Mean and standard deviation values are attained in normalization process, and these values are normalized to distinguish normal from abnormal states. Second, there is a need to binarize all data because of the structure of a DBN. DBN only uses to binarized data. The binarized data have great effects of accuracy to establish the performance before classification (Masmoudi et al, 2013). After normalization, most data are shown near the value of zero, assuming low frequency when the values are far from zero. In this process, the data are quantized and assigned with their respective bits—to be compressed into either 8 bits or 10 bits. The error rates of 8 bits and 10 bits are as follows. The research assumed that 8 bits and 10 bits are enough to conduct the performance appropriately.

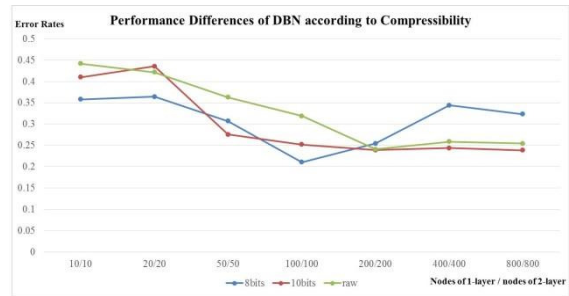


Figure 12. Performance differences of DBN according to compressibility

As a result, 10 bits of compressibility have better performance compared to others. Therefore, this research applied 10 bits of compressibility of data into fault detection and diagnosis.

3.2 Classifier: Deep Belief Network

“Classification” is the process of fault detection and diagnosis when the data of the sensors with Modelica are attained. Among the various classification methods such as support vector machines (SVM), k-nearest neighbors (K-NN) and so on based on other researches, this research used the “deep learning method,” which is one of the most popular machine learning methods available. Because there is no information on normal or abnormal data of sensors with Modelica using a classifier, this research made use of a deep belief network (DBN) as a classifier, which uses various data to assess whether the data are normal or abnormal. Before using the DBN classifier, durations (number of iterations) and structures are needed to be determined. Tests are repeatedly carried out to attain the appropriate the number of iterations and structures of the DBN. The results of such tests are as follows.

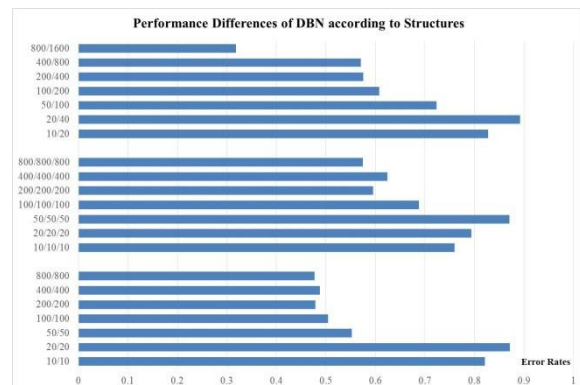


Figure 13. Performance differences of DBN according to structure

The structures of DBN are attained from 2-layer models to change the number of nodes of layers. As in the first row of Figure 11, 800/1,600 means that the test conducted 800 nodes of the first layer and 1,600 nodes of the second layer.

Also, this research conducted the number of iterations that is suitable to achieve optimal values. The results of 300 and 1000 as the number of iterations are as follows.

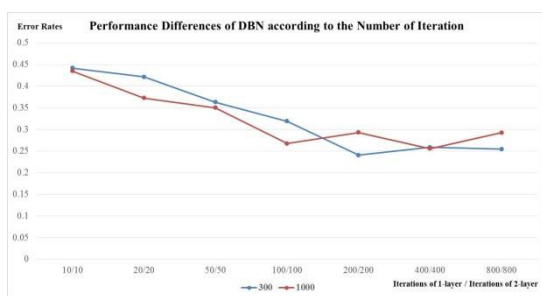


Figure 14. Performance differences of DBN according to the number of iteration

As the results, 800–1,600 of 2-layers and 300 as the number of iterations are appropriate to the application of the DBN. In this research, 2-layers are enough to accomplish the performance due to short-time calculations.

In Section 2, three faults are appointed and the results of fault detection and diagnosis are shown in Table 2.

Faults	Detection and Diagnosis (%)
Supply Fan	81%
Valve	85%
Heat Exchanger	99.16%

Table 2. Results of FDD

3.3 Post-processing

“Post-processing” is the process that increases the rates of detection and diagnosis, and this is where the deferral rate is set. “Deferral rate” means the instance in which no decision is made as the

judgment does not ensure whether which is normal or abnormal. If there is a deferral rate of 11%, 89% of the data are used to determine the test. The results of the detection and diagnosis rates that take the deferral rates into consideration are shown in the Table 3.

Faults	Detection and Diagnosis (%)	Deferral Rate (%)
Supply Fan	95%	11%
Valve	95%	31%
Heat Exchanger	99.16%	0%

Table 3. Results of FDD considering deferral rates

4 Conclusion

Various researches on fault detection and diagnosis in HVAC systems have been published; however, these studies lack inclusion of data on real sensors and disposal of noises. In addition, actual application of data measurement of simple correlation is difficult despite the use of complicated models and methods. To overcome such limitation, this research used the machine learning method, and verified fault detection and diagnosis using specific data with Modelica like as a real AHU of HVAC system. The accuracy of the results of this study’s fault detection and diagnosis was given an approximate score of above 95%. With this, it is necessary to verify actual data from real buildings for future studies.

Acknowledgments

The research of this paper is supported by Hyundai Engineering and Construction Company.

References

Massieh Najafi, David M, Auslander, Peter L. Bartlett, Philip Haves, Michael D, Shon, Modeling and measurement constraints in fault detection and diagnostics for HVAC systems, 2010
 Zhimin Du, Singiao Jin, Yunyu Yang, Wavelet neural network based fault diagnosis in air handling unit,

2008

- J.M.House, H.Vaezi-Nejad, J.M.Whitcomb. An expert rules set for fault detection in air handling units, ASHRAE Trans. 107, 858-871, 2001
- J.Schein, S.T. Bushby, N.S. Castro, J.M. House, A rule-based fault detection method for air handling units, Energy Build. 38, 1485-1492, 2006
- T.I.Salsbury, R.CDimond, Fault detection in HVAC systems using model-based feed forward control, Energy Build. 33, 403-415, 2001
- B. Yu, A.H.C. van Passen, S. Riahy, General modeling for model-based FDD on building HVAC systems, Simulat, Pract., Theory 9(6-8), 387-397, 2002
- ASHRAE. Sequences of Operation for Common HVAC Systems. ASHRAE, Atlanta, GA, 2006.
- Deru M., K. Field, D. Studer, K. Benne, B. Griffith, P. Torcellini, M. Halverson, D. Winiarski, B. Liu, M. Rosenberg, J. Huang, M. Yazdanian, and D. Crawley. DOE commercial building research benchmarks for commercial buildings. Technical report, U.S. Department of Energy, Energy Efficiency and Renewable Energy, Office of Building Technologies. 2009.
- Modelica Buildings Library developed by LBNL. Modelica library for building energy and control systems. <http://simulationresearch.lbl.gov/modelica>
- Top of Alabama Regional Council of Governments. TARCOG: Mathematical models for calculation of thermal performance of glazing systems with our without shading devices, Technical Report, Carli, Inc. 2006.
- Masmoudi, Y, Turkay, M, Chabchoub, H, A binarization strategy for modelling mixed data in multigroup classification, Advanced Logistics and Transport, 345-353, 2013
- Yuebin Yu, Denchai Woradechjumbo, and Daihong Yu (2014): A Review of Fault Detection and Diagnosis Methodologies on Air-handling Units. Energy and Buildings, 82:550–562, 2014.
- Zhengwei Li, Adaptable, scalable, probabilistic fault detection and diagnostic methods for the HVAC secondary system. Dissertation. Georgia Institute of Technology. 2012.
- Zhimin Du, Bo Fan, Xingqiao Jin, and Jinlei Chi (2014): Fault Detection and Diagnosis for Buildings and HVAC Systems Using Combined Neural Networks and Subtractive Clustering Analysis. *Building and Environment*, 73:1–11, 2014.
- Geoffrey E. Hinton, Simon Osindero, Yee-Whye The, A fast learning algorithm for deep belief nets, Neural Computation, 1527-1554, 2006

Formal Requirements Modeling for Simulation-Based Verification

Martin Otter¹, Nguyen Thuy², Daniel Bouskela², Lena Buffoni³, Hilding Elmqvist⁴, Peter Fritzsön³, Alfredo Garro⁵, Audrey Jardin², Hans Olsson⁴, Maxime Payelleville⁶, Wladimir Schamai⁷, Eric Thomas⁶, Andrea Tundis⁵

¹Institute of System Dynamics and Control, DLR, Germany, Martin.Otter@dlr.de

²EDF, France, {Daniel.Bouskela,Audrey.Jardin,N.Thuy}@edf.fr

³PELAB, Linköping University, Sweden, {Lena.Buffoni,Peter.Fritzsön}@liu.se

⁴Dassault Systèmes AB, Sweden, {Hilding.Elmqvist,Hans.Olsson}@3ds.com

⁵DIMES, University of Calabria, Italy, {Alfredo.Garro,Andrea.Tundis}@unical.it

⁶Dassault Aviation, France, {Eric.Thomas,MP}@dassault-aviation.com

⁷Airbus Group Innovations, Germany, Wladimir.Schamai@airbus.com

Abstract

This paper describes a proposal on how to model formal requirements in Modelica for simulation-based verification. The approach is implemented in the open source Modelica_Requirements library. It requires extensions to the Modelica language, that have been prototypically implemented in the Dymola and OpenModelica software. The design of the library is based on the Formal Requirement Modeling Language (FORM-L) defined by EDF, and on industrial use cases from EDF and Dassault Aviation. It uses 2- and 3-valued temporal logic to describe requirements.

Keywords: requirements, verification, physical systems, 3-valued logic, temporal logic.

1 Introduction¹

1.1 Overview

To ensure the proper operation of complex physical systems such as power plants, aircraft or vehicles, requirements are issued all along the system's lifecycle: from the preliminary design phase to the operation phase. Typically, the requirements capture the spatiotemporal and quality of service conditions that a system should fulfill. They may be quite complex and numerous. Testing the compliance of the system with the requirements may be quite challenging, due to the many items that should be examined and verified for a given test scenario, and the number of test scenarios to be considered to have a satisfying verification coverage.

This paper tries to improve the current situation, by (a) providing the open source library Modelica_Requirements to define and model requirements in a formal way using 2- and 3-valued linear temporal logic (LTL); (b) associating requirement models with behavioral models; (c) testing whether the defined

requirements are violated by the system design currently studied when the underlying behavioral models are *simulated*. This approach requires extensions to Modelica, that have been prototypically implemented in Dymola (Dassault Systèmes, 2015) and in OpenModelica (Open Source Modelica Consortium, 2015). The library has been tested and can be used by both of these Modelica simulation environments.

The main purpose of this approach is to check formally defined requirements by *simulation*. It is *not* intended to perform formal model verification by model checkers as done by tools such as NuSMV², SPIN³, Prover Plug-in⁴ for discrete systems or SpaceX⁵, KeYmaera⁶ for hybrid systems. For example, a differential-algebraic equation system may be solved numerically to compute a pressure p in a pipe, and the requirement is formulated as $p \geq p_{\text{cavitate}}$. Model checkers for discrete systems cannot be used in this case, and verification tools for hybrid systems can only handle simple sets of differential and discrete equations, but not large models of industrial applications like power plants or aircraft.

1.2 State-of-the-art to Define Requirements

The standard in industrial applications is still to define requirements in natural language in textual form. As a typical example see the requirements for electrical systems in US military aircraft MIL-STD-704F (Department of Defense, 1984). Such specifications are defined in reports by using for example Microsoft Word, or with dedicated tool support. The latter especially to get support for collaboration, traceability, coverage analysis of textually defined requirements. Moreover, visual modeling languages for system

¹ This section uses material from the internal reports (Bouskela et al. 2015) and (Otter et al., 2014).

² NuSMV: <http://nusmv.fbk.eu/>

³ SPIN: <http://spinroot.com/spin/whatispin.html>

⁴ Prover Plug-in: http://www.prover.com/products/prover_plugin/

⁵ SpaceX: <http://spacex.imag.fr/>

⁶ KeYmaera: <http://symbolaris.com/info/KeYmaera.html>

engineering are very common, such as SysML⁷, a general-purpose modeling language for systems engineering applications, that defines requirement and parametric diagrams for supporting the modeling of system properties. In particular, requirement diagrams provide constructs and mechanisms to express and compose system requirements, as well as to allocate them to system components; parametric diagrams can be used for supporting performance analysis and quantitative assessment. There are a number of tools in this area, for example: Rational DOORS from IBM⁸, Reqify from Dassault Systèmes⁹, OSRMT (GPL2)¹⁰, formalmind Studio (free)¹¹. The most important xml-based exchange format seems to be ReqIF (OMG, 2013).

Defining and processing requirements *formally* is an area of active research. The exploited mathematics uses propositional logic, temporal logic, set theory and others; see for example (Baier and Katoen, 2008; Lamport, 2015). There are many publications, but the pure mathematical notation is quite far away from a language an engineering practitioner would be able to use.

For electronic circuit design, there is a proposal for an Analog Specification Language (ASL) by (Steinhorst and Hedrich, 2009), with a detailed proposal of language elements and some examples. In (Schamai, 2013) the idea for formalizing a natural-language requirement into a requirement violation monitor is presented. In runtime verification, monitors are expressed in some variant of linear temporal logic expressions and to generate efficient code for the actual monitors (Leucker and Schallhart, 2009).

The SIMULINK toolbox “Verification and Validation”¹² from MathWorks is used to define formal requirements in SIMULINK and to automatically test and verify requirements by *simulation*. In the master thesis (Tunnat, 2011) the toolbox has been applied to an aircraft system. Figure 1 is an example from this thesis that shows the essential elements (in the thesis a script was implemented for the report generator of SIMULINK, that combines the textual description in a Word file with the screen shot of the formal definition in Stateflow): The *Detector* delays and/or synchronizes Boolean signals, the *Implies* block is the logical implies operator of Boolean algebra, and *Assertion* expects that its input is always true and triggers a requirements failure if this is not the case. Note, that requirements are defined with 2-valued logic.

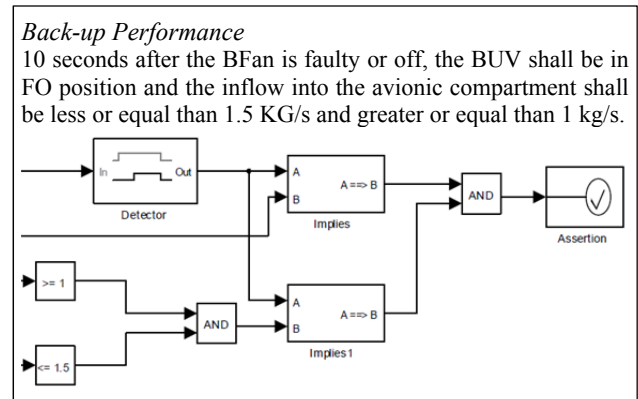


Figure 1. An example of a requirement definition with the SIMULINK toolbox “Verification and Validation”. Text and figure from (Tunnat, 2011).

1.3 Modelica_Requirements Prerequisites

In two recent ITEA projects, EUROSYSLIB¹³ and OPENPROD¹⁴, part of the research was devoted to how to model requirements in Modelica. The EUROSYSLIB results are reported in (Jardin et al., 2011) and resulted in conceptual work and a prototype Modelica library. The OPENPROD results are partially reported in (Schamai, 2013).

In the ITEA MODRIO¹⁵ project, EDF developed a complete concept for a central industrial scenario: First defining the requirements for a system, then performing an architectural design that shall comply with the requirements and finally evaluating and fine-tuning the architectural design with behavioral models (Bouskela et al., 2015). Furthermore, EDF developed the special language FORM-L (Thuy, 2014) to describe requirements in a formal way but close to the (textual) notation used by system designers. EDF evaluated and refined the language on a larger benchmark example (Thuy, 2013). In (Garro et al., 2014) it was systematically evaluated how to map FORM-L language elements and ideas to Modelica. The above work, including new investigations of Dassault Aviation, finally resulted in the Modelica_Requirements library described in the following sections.

2 Modelica_Requirements Library

The top-level view of this library is shown in Figure 2. The library has about 200 model and block components and about 50 functions. It is provided under the Modelica License 2, and can therefore be used in commercial applications without essential restrictions. The most important sub-libraries are discussed in the following sub-sections.

⁷ SysML: <http://www.omgsysml.org>

⁸ DOORS: <http://www-03.ibm.com/software/products/en/ratidoor>

⁹ Reqify: <http://www.3ds.com/products-services/catia/capabilities/requirements-engineering/reqify/>

¹⁰ OSRMT: <http://sourceforge.net/projects/osrmt/>

¹¹ formalmind studio: <http://formalmind.com/studio>

¹² SIMULINK toolbox “Verification and Validation”: <http://www.mathworks.com/products/simverification>

¹³ EUROSYSLIB: <https://itea3.org/project/eurosyslib.html>

¹⁴ OPENPROD: <https://itea3.org/project/openprod.html>

¹⁵ MODRIO: <https://itea3.org/project/modrio.html>

2.1 Two- and Three-valued Logic

Defining elements with formal logic requires defining an appropriate data type. All programming languages support two-valued logic. In Modelica, the data type `Boolean` is used for this purpose. FORM-L uses three-valued logic. Also, several publications in this area suggest using three-valued logic, see for example (Schamai, 2013).

Important reasons for using three-valued logic are:

(1) In certain situations it is not possible to state whether a property is violated/false or satisfied/true. For example the FORM-L operator

```
during(condition, check)
```

is defined as: “As long as the `condition` is `true`, `check` must be `true`”. However, what return value should be used, when `condition` is not `true`? (e.g. when the component to be checked is not “in operation”). This case is not defined and therefore the operator should neither return `false` nor `true`, but undefined. There are also operators where during a first time range, the return value of the operator is not defined and therefore the best meaningful value to return is undefined. With two-valued logic the user has to either return two Booleans to describe this situation, or somehow select a value `false` or `true` in such cases. The problem is that logical expressions that depend on such an arbitrarily selected value may make a required property violated or satisfied, although in reality it is undecided and this may either give an overly optimistic or an overly pessimistic view.

(2) Simulations with requirement models should determine whether a required property is violated. A simulation may, however, not evaluate a defined requirement model (e.g. if only simulations are performed where the model to be checked is not “in operation”). With three-valued logic this situation can be indicated by, e.g. the value `undecided`. With two-valued logic it cannot be stated that a simulation did not test all required properties, and when the simulation run returns with “all required properties satisfied”, this might be too optimistic or simply wrong.

Three-valued logic has the following drawbacks:

(1) There are several types of three-valued logic definitions, such as Kleene's, Lukasiewicz's,

Bochvar's and other logics (Lukasiewicz, 1920; Bochvar, 1937; Breuer, 1972; Rescher, 1969). Some operators, like “`or`” and “`and`” are identical in the different schemes, but the `implies(a,b)` operator is not. For an user it is not obvious which three-valued logic is used in a system and what the consequences are.

(2) Modelica has already many operators and functions for two-valued logic and also users will have many models utilizing two-valued logic. If three-valued logic alone were to be used for requirements modeling, then a large amount of existing code could not be reused.

It is clear that two-valued logic must be supported in order to use existing code and to support the well-known view of the user on logical expressions, as well as language elements such as `if/else` or `while`. On the other hand, two-valued logic alone has disadvantages for requirements modeling as sketched above. For these reasons, in the Modelica_Requirements library two-valued logic, as well as a *restricted form of three-valued logic* is used. The three-valued logic is defined by enumeration `Property` (in sub-library `Types`):

```
type Property = enumeration(Violated,
                             Undecided,
                             Satisfied);
```

Only functions and blocks with three-valued logic input and/or output arguments are used where the semantics can be defined mathematically in a uniquely accepted way that is also natural and obvious for the user. For example, the function

```
during(condition, check)
```

is provided with `Boolean` input arguments `condition` and `check`, and a `Property` return value. On the other hand, a function `implies(..)` with three-valued logic input/output arguments is not provided because different types of three-valued logics are in use and the result value is not obvious for a user. Also cast functions from `Boolean` and `Integer` to `Property` and from `Property` to `Boolean` and `Integer` are provided. The mapping from `Property` to `Boolean` is not unique, because it is not obvious how to map the value “`Undecided`” to a `Boolean`. This issue is resolved by requiring users to specify the mapping with a second input argument:

```
Property p = ...;
Boolean b;
equation
  b = PropertyToBoolean(p, undecided=true);
```

To simplify the view for the user, most functions and blocks have at most one input argument and/or one output argument of type `Property`. The only exceptions are the 3-valued blocks to model the `or`, `and`, `not` operators in 3-valued logic, for which a

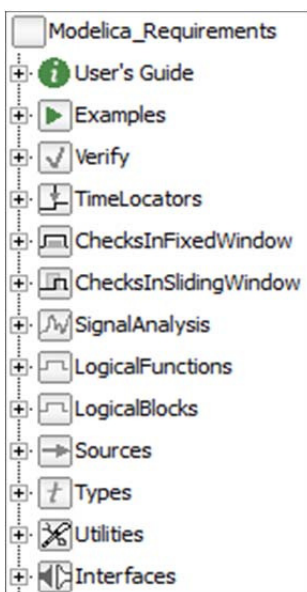
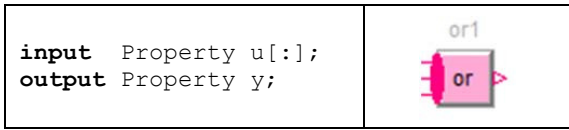


Figure 2. Modelica_Requirements library.

commonly accepted unique definition exists. For example, the `LogicalBlocks.PropertyOr` block is defined as (in the next figure, three connection lines have been drawn to instance “or1”):



where $y = u[1] \text{ or } u[2] \text{ or } u[3] \text{ or } \dots$, and using the truth-table (here for two inputs):

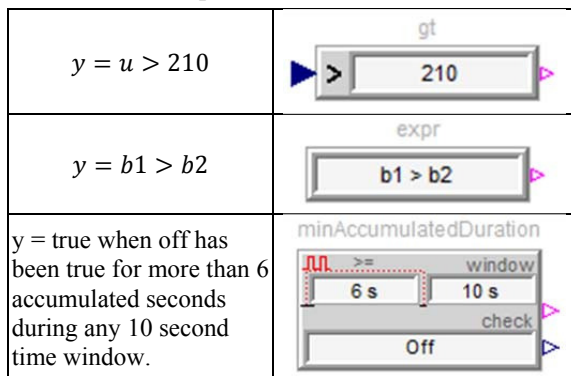
$u[1] \text{ or } u[2]$	<i>Violated</i>	<i>Undecided</i>	<i>Satisfied</i>
<i>Violated</i>	Violated	Undecided	Satisfied
<i>Undecided</i>	Undecided	Undecided	Satisfied
<i>Satisfied</i>	Satisfied	Satisfied	Satisfied

2.2 Graphical Layout

It is expected that the `Modelia_Requirements` library is utilized by users, such as system architects, without requiring that they be simulation specialists. For this reason an effort was made to improve the usual graphical appearance of models/blocks (within the limitations of Modelica). The following principles are used:

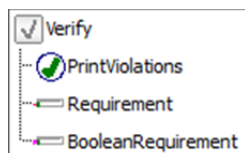
- (1) All entries of a parameter menu are displayed in the icon, in order that it not be necessary to inspect the menu to understand the parameterization (as a consequence, a menu, and therefore a block, must be simple and can have at most 3 or 4 input fields).
- (2) All such menu entries are defined as “input fields” to make visually clear that the user can provide values (see examples below).
- (3) The instance name is displayed above the icon, but in light grey, in order that it not disturbs the layout too much. One could remove the instance name completely from the icon, but it is then no longer so easy to select plot variables by name.

Here are some examples:



2.3 Definition of Required Properties

In sub-library `Verify` blocks are present to (a) define that a `Property` or `Boolean` signal is a



required property and (b) to print a log summary after a simulation (see figure). An example for the usage of block `Requirement` is shown in the next figure:

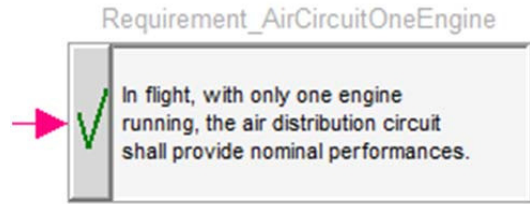


Figure 3. Example on how to define a required property.

The left hand arrow is an input signal of type `Property`. In the icon, the content of parameter `text` is displayed that should contain a textual description of the required property. For this, a new annotation “`AutoLineBreak`” is proposed that displays a `String` parameter in the icon with automatically selected line breaks (so that the text with a given font, here 8pt, is displayed within the surrounding box):

```
parameter String text annotation(AutoLineBreak=true);
```

The `Requirement` block monitors its property input over a simulation and computes its status at the end of the simulation run:

- *Requirement is violated:*
Input is `Violated` at least once.
- *Requirement is untested:*
Input is `Undecided` for the complete simulation run
- *Requirement is satisfied:*
Input is `Satisfied` at least once, and is never `Violated`.

Determining this status is more difficult than one would expect, because during event iteration a requirement may become temporarily violated, but at event restart the requirement may no longer be violated. To avoid false messages of this type, one has to determine whether a requirement is violated at event restart. This is achieved with the following Modelica code:

```
when not terminal() and change(property) then
  if not pre(atLeastOneFailure) and
    property == Property.Violated then
    atLeastOneFailure = true;
    firstFailureTime = time;
  elseif pre(atLeastOneFailure) and
    time <= firstFailureTime and
    (property==Property.Satisfied or
     property==Property.Undecided) then
    atLeastOneFailure = false;
    firstFailureTime = startTime - 1;
  end if;
end when;
```

The `when`-clause becomes active, whenever `property` changes its value. If `property` became `Violated` the first time, this is marked with `atLeastOneFailure = true`. If `property` is changing at the current event iteration, determined by `time <= firstFailureTime`

(time is not changing at an event, and therefore this expression will be true at the same event instant), again a check is made whether `property` is no longer Violated. In this case, `atLeastOneFailure` is set back to `false`.

The information about the instance name of the requirement, the requirement text and its status are stored on a log file in textual format. This log file could be processed after the simulation run for example by a script. Additionally, the user can drag the block `PrintViolations` to the top level of his/her model, see Figure 4.



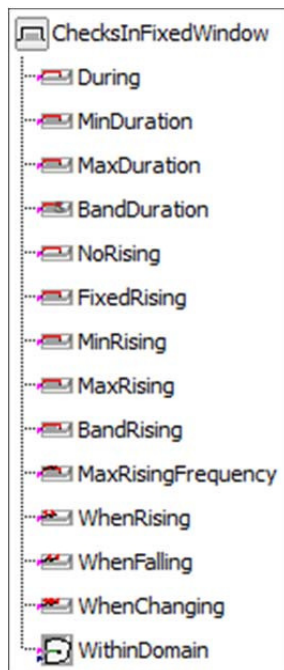
Figure 4. Defining requirement status log (left figure: icon; right figure: parameters of the block)

This block prints a detailed summary of the status of all requirements to the output window. The output can be configured, see right side of Figure 4. Furthermore, the “satisfaction” factor, that is the percentage of requirements with status = `Satisfied`, are dynamically displayed in the icon (see left side of Figure 4) and stored in the result file, to give a quick overview about the requirement status.

2.4 Checks in Fixed Windows

In sub-library `ChecksInFixedWindow` (see figure to the right) blocks are present that determine whether a particular property is fulfilled or not in a *given time window*: Whenever the Boolean input `condition` is true, the property is checked, otherwise the property is not checked (and the output is set to `Undecided`). Properties that can be checked are for example, that input `check`

- must be true for a minimum and/or a maximum duration,
- must have a minimum and/or a maximum number of rising edges.



For example, with block `MaxRising`, see Figure 5, it is stated that the number of rising edges of `check` is limited during every true `condition` phase. The left input arrow is `condition` and the lower input field is `check = engineStart`, so that at most three tries of `engineStart` (becoming true) are allowed in the

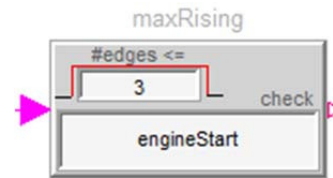


Figure 5. Example for `MaxRising` block.

start phase (`condition = true`).

In a first design, `check` was not provided by an input field, but by an additional input connector to the left. In larger use cases, like the EDF Backup Power Supply (Thuy 2013), it turned out that the diagram layer of the requirement models became hard to understand due to the many connection lines. This issue could be reduced by using an input field with a name for the `check` signal instead of a connector.

The implementation of most of the blocks in this sub-library is straightforward. For example, the `MaxRising` block is implemented as¹⁶:

```

initial equation
  countRising = 0; // number of rising edges
  y = if condition then Property.Satisfied
      else Property.Undecided;

equation
  when condition then
    countRising = 0; y = Property.Satisfied;
  elseif condition and check then
    countRising = pre(countRising) + 1;
    y = if countRising <= nRisingMax then
        Property.Satisfied else Property.Violated;
  elseif not condition then
    countRising = 0; y = Property.Undecided;
  end when;
    
```

A typical simulation result is shown in the next figure:

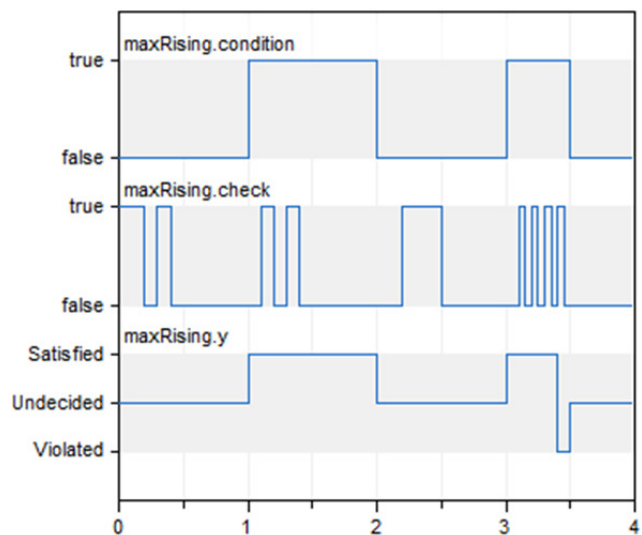


Figure 6. Simulation result for example of Figure 5.

Note, that between 3.4s .. 3.5s the output is `Violated`, because there have been 4 rising edges of `check`.

¹⁶ Rising edges are not counted at the time instant when `condition` becomes true or when it becomes false.

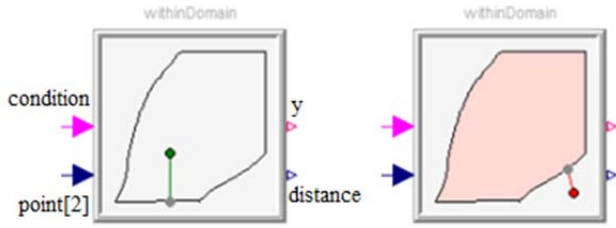


Figure 7. Example for `WithinDomain` block (left figure: point is within the domain, right figure: point is outside the domain)

`WithinDomain` is a more complicated block, see left part of Figure 7. This block defines a domain with a polygon and the requirement is that the input point (a vector of size 2 defining the x- and y-coordinate of the point) must be within this domain. For example, in a passenger aircraft the “time to complete a cabin pressure change” (x-coordinate) and the “cabin altitude rate of change” (y-coordinate) must be within a given 2-dimensional domain that can be described by the `WithinDomain` block.

The actual polygon is displayed in the icon, together with the point (= green circle) and the nearest distance of the point to the polygon. After a simulation run, a diagram animation shows the actual status. In the right part of Figure 7 the point is outside of the polygon and then the domain and the point is displayed in red. Output y is

- Undecided if `condition = false`,
- Satisfied if `condition = true` and the point is within the polygon and
- otherwise it is `Violated`.

Displaying the polygon, the point and the distance in the icon is performed with the standard Modelica annotation `DynamicSelect(.)` that allows an element in an icon to be displayed dynamically. Determining the distance of a point to a polygon is a standard task in computer graphics. In the block a pure Modelica implementation is used. The relationships of one line of the polygon are displayed in Figure 8:

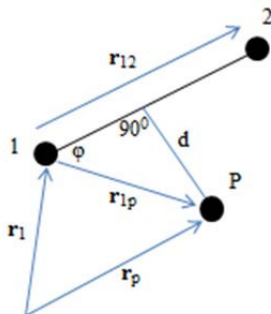


Figure 8. Relationships between one polygon line 1→2, point P and the closest distance d of P to this line.

The corresponding equations are:

$$\begin{aligned} \mathbf{r}_{12} &= \mathbf{r}_2 - \mathbf{r}_1 \\ \mathbf{r}_{1p} &= \mathbf{r}_p - \mathbf{r}_1 \\ \mathbf{r}_d &= \mathbf{r}_1 + \lambda \cdot \mathbf{r}_{12} \end{aligned} \quad (1)$$

The cosine φ of the angle between vectors \mathbf{r}_{12} and \mathbf{r}_{1p} can be either computed with the relationships in a triangle, or with the dot-product, where λ with $0 \leq \lambda \leq 1$ characterizes the point \mathbf{r}_d on the line with the shortest distance to P:

$$\cos \varphi = \frac{\lambda \cdot |\mathbf{r}_{12}|}{|\mathbf{r}_{1p}|} = \frac{\mathbf{r}_{12} \cdot \mathbf{r}_{1p}}{|\mathbf{r}_{12}| \cdot |\mathbf{r}_{1p}|} \quad (2)$$

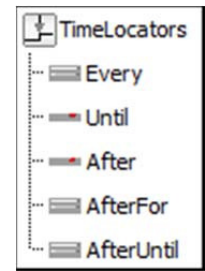
and therefore

$$\begin{aligned} \lambda &= \max\left(\min\left(\frac{\mathbf{r}_{12} \cdot \mathbf{r}_{1p}}{|\mathbf{r}_{12}| \cdot |\mathbf{r}_{12}|}, 1\right), 0\right) \\ d &= |\mathbf{r}_{1p} - \lambda \cdot \mathbf{r}_{12}| \end{aligned} \quad (3)$$

Equations (3) are applied on every segment of the polygon, and the smallest distance d to all of the segments is selected. Another algorithm computes whether point P is within or outside of the polygon and d is set to a negative value if P is outside of the polygon.

2.5 Time Locators

The condition inputs of the blocks from sub-library `ChecksInFixedWindow` are Booleans that may originate from quite different sources. Due to the importance of these conditions, sub-library `TimeLocators` provides often occurring continuous-time locators, that are temporal operators to define the condition interval of interest (see figure to the right).



The outputs of these blocks are Booleans that can be used directly as condition inputs to the blocks of `ChecksInFixedWindow`. FORM-L (Thuy, 2014) has also more complex type of time locators. It is planned to support them as well.

Modelica does not have an “Event” data type. Instead, rising or falling edges of Boolean variables are used to define a time instant of interest that might be described in other modeling systems by an “Event”. The following blocks of sub-library `TimeLocators` are currently available:

- `Every`: Output is true during every interval for a defined duration.
- `Until`: Output is true until first rising edge of input.
- `After`: Output is true after first rising edge of input.
- `AfterFor`: Output is true after rising edge of input for a defined duration.
- `AfterUntil`: Output is true, after rising edge of input 1 until the rising edge of input 2

The implementation of these blocks is straightforward. In Figure 9 an example from (Thuy, 2013) is shown using block `AfterUntil`. This example concerns the generator of a Backup Power system:



Figure 9. Example for block AfterUntil.

The generator can signal several events (= rising edges of Boolean signals), including `eStart` (it has started) and `eStop` (it has stopped). Therefore, Figure 9 defines the time periods where the generator is running. For these time periods required properties might be defined with blocks from sub-library `ChecksInFixedWindow`.

The `AfterUntil` block is implemented as:

```

input Boolean u1 "Boolean input 1 (after)";
input Boolean u2 "Boolean input 2 (until)";
output Boolean y "= true, after rising edge of u1
                until rising edge of u2";

initial equation
y = u1;
equation
when u1 then
y = true;
elsewhen u2 then
y = false;
end when;
    
```

A simulation result is shown in Figure 10: The generator is running (`afterUntil.y = true`) between two rising edges of `eStart` and `eStop`.

2.6 Checks in Sliding Windows

In sub-library `ChecksInSlidingWindow` (see figure to the right) blocks are present that determine whether a particular property is fulfilled or not in a *sliding time window*. For example, if a sliding time window has size T and t is the actual time instant, then in every time range $[t - T, t]$ the property must be fulfilled.

Evaluating a property in a sliding time window requires storing the values of the relevant signals in a buffer that covers “essential” signal values in the past at least up to time $t - T$, and operating on this buffer. For Boolean signals a buffer has been

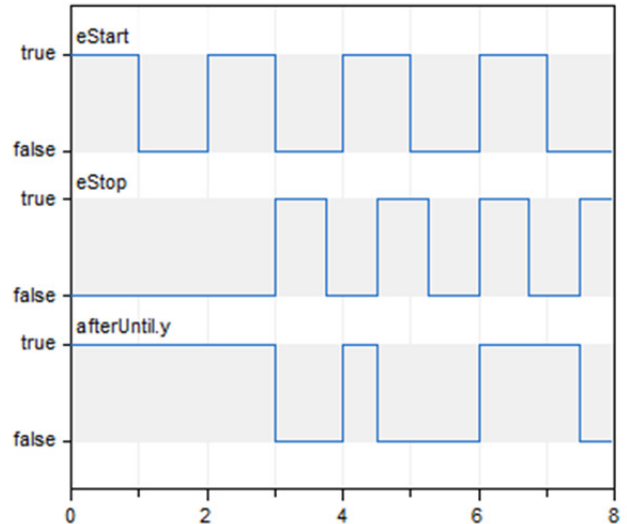
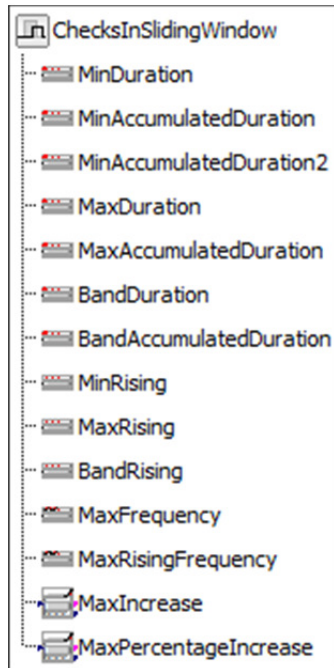
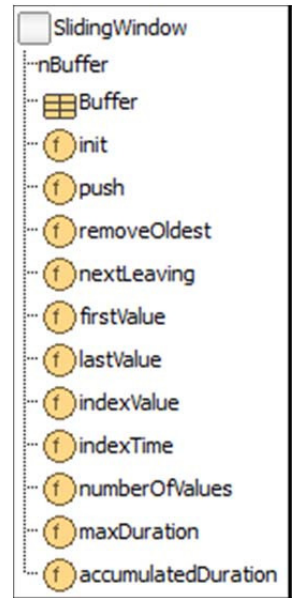


Figure 10. Simulation result for example of Figure 9.

designed which is available as `Internal.SlidingWindow` (see the figure on the right). This is a package consisting of a record `Buffer` in which past values are stored and a set of functions operating on this record. The current implementation is a pure Modelica implementation to gain experience and figure out the right function interfaces. Since a “memory” is needed that is passed between Modelica functions, the size of this memory must be fixed at compilation time and the complete buffer must always be copied, once an element in this buffer is changed. It is planned to replace this implementation by a C-implementation with a Modelica `ExternalObject` to get rid of these restrictions.



The `SlidingWindow` buffer package is basically a queue where elements with a time stamp t are inserted at the top and elements with a time stamp older than $t - T$ are removed at the bottom. The memory of the queue is defined as (where `nBuffer=20` is a defined constant):

```

record Buffer "Memory of sliding window"
Modelica.Slunits.Time T "Length of sliding time win.";
Modelica.Slunits.Time t0 "Time instant where sliding
                        time window starts";
Modelica.Slunits.Time t[nBuffer] "Time instants";
Boolean b[nBuffer] "Values at corresp. time instants";
Integer first "Index of first element in buffer";
Integer last "Index of last element in buffer";
Integer nElem "Number of elements in the buffer";
end Buffer;
    
```

Some of the functions operating on this buffer are sketched at hand of block `MinAccumulatedDuration2`, see Figure 11.

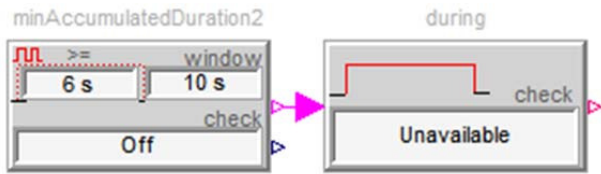


Figure 11. Example for `MinAccumulatedDuration2`

This example models the following requirement from (Thuy, 2013):

When the MPS (Main Power Supply system) is switched off, signaled by Boolean `Off`, then the MPS must be declared `Unavailable` when it has been off for more than 6 accumulated seconds during any 10 seconds time window.

This is achieved in the following way: Component `minAccumulatedDuration2` outputs `true`, if in any time window of length 10 s variable `Off` was accumulated `true` for at least 6 s. This signal is the input to component `during` which requires that whenever the input is true, variable `Unavailable` must be true as well. In that case the block outputs `Satisfied`. If the input of `during` is true and `Unavailable = false`, the requirement is clearly violated and the `during` block outputs `Violated` (if the input is false, the block outputs `Undecided`).

Block `MinAccumulatedDuration` outputs a Property whereas `MinAccumulatedDuration2` outputs a Boolean. The difference is only during the initial phase $t < t_0 + T$ where the first block returns `Undecided` if the property is violated, and the second returns `false`. The `MinAccumulatedDuration2` block is implemented in the following way

```
import Modelica_Requirements.Internal.SlidingWindow.*;
parameter Modelica.SIunits.Time window;
parameter Modelica.SIunits.Time lowerLimit;
input Boolean check(start = false);
output Boolean y "= true if property satisfied";
output Real accDuration;
protected
  Buffer buffer "Buffer for sliding window";

initial equation
  buffer = push(init(T,time), time, check);
  pre(check) = check;

equation
  when change(check) then
    buffer = push(pre(buffer), time, check);
  end when;
  accDuration = accumulatedDuration(buffer, time, check);
  y = accDuration >= lowerLimit;
```

The `Buffer` functions have the following tasks:

- `init(T,time)` returns an instance of `Buffer` and initializes it with the length of the sliding time window `T` and the initial time instant `time`.
- `push(init(T,time), time, check)` generates and initializes a `Buffer` and stores one element (= the initial time instant and the value of `check`) in the buffer. At the same time, this call removes values from the buffer that have a time stamp older than `time - T`. The function returns a copy of the buffer.
- The code


```
when change(check) then
  buffer = push(pre(buffer), time, check);
end when;
```

 is executed whenever `check` changes its value (and at that time instant an event occurs). The function call stores the actual time instant and the value of `check` in the buffer from the last event instant and removes older values from the buffer. The updated buffer is then returned at the actual event instant.
- The code


```
accDuration = accumulatedDuration(buffer, time, check);
```

 is executed during *continuous-time integration*, that is whenever the integrator requires a model evaluation. The function call `accumulatedDuration(..)` computes the accumulated time duration where the values of `check` in the buffer have been true during the time window `time - T` and returns this value. The third argument `check` of this function call is usually ignored, but is used if the buffer is empty.
- The code


```
y = accDuration >= lowerLimit;
```

 triggers a state event when the accumulated time duration crosses its limit and `y` changes its value from `false` to `true` or from `true` to `false` depending on the crossing direction.

2.7 Utility Functions and Blocks

Besides of the already discussed core blocks, the `Modelica_Requirements` library has also quite a lot of utility functions and blocks that might be useful to formally define a requirement:

Sub-library `SignalAnalysis` consists of blocks to compute exact or approximate derivatives, an integrator that can be controlled by a trigger signal, a moving average filter, and other blocks.

Sub-library `LogicalBlocks` provides blocks to convert between Boolean, Integer and Property signals, comparing Real signals as well as logical operators (`not`, `or`, `and`) on Property signals. Some of these blocks are also available in the `Modelica Standard Library`. However, since they seemed to be often needed for requirements modeling, they have been provided additionally with the graphical layout used for the `Modelica_Requirement` blocks.

When textually modeling or when implementing blocks, a set of useful functions for 2- and 3-valued

logic have been collected in sub-library `LogicalFunctions`. Some of these functions are motivated by the FORM-L language and provide set-like functionality on Modelica vectors. For example function `exist(.)` has a Boolean input vector and returns true if at least one element of this vector is true. In combination with Modelica's reduction expressions, quite powerful compact formulations are possible, as shown in the next example:

```
// Define a set of pumps
Pump pumps[3] = {Pump(isActive=time < 1 or
                    time > 2 and time < 3),
                Pump(isActive=time < 0.5 or time > 2.5),
                Pump(isActive=time > 1.5 and time < 1.9)};

// At least one pump must be active all the time
Boolean atleastOnePumpActive =
    exists({p.isActive for p in pumps})
```

Sub-library `Examples` contains a large set of examples to demonstrate and assess the components of the library. Every component of the library is present in at least in one example (so class coverage is 100 %).

There is also a growing set of application specific examples that can be used as templates in actual projects. For example, sublibrary `Modelica_Requirements.Examples.AircraftRequirements` contains typical requirement definitions used in aircraft systems:

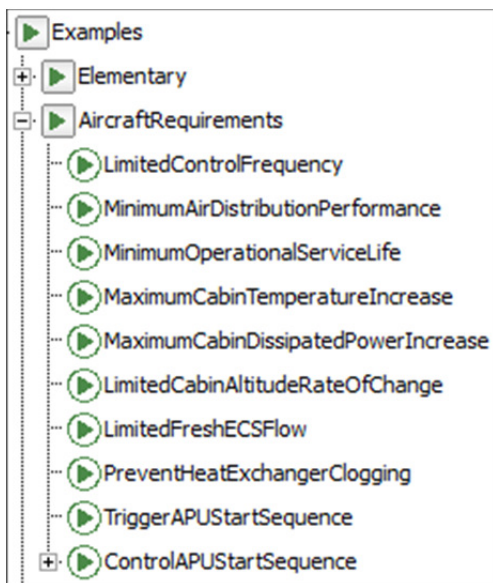


Figure 12. Sub-library of aircraft specific requirements from Dassault Aviation.

Every example contains a short definition of the requirement (as it is typically present in design documents), the corresponding Modelica model to verify the requirement together with some simple test signals. For example, the requirement “*In the cabin area, the temperature increase should not exceed 3°C per hour.*” is verified with the following model (the input is cabin temperature as function of time defined in a table):

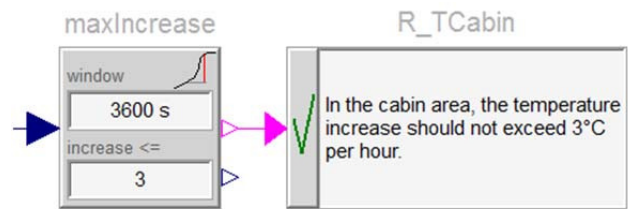


Figure 13. An aircraft requirement to assess the limited allowed temperature increase in the cabin area.

3 Textual Definition of Requirements

In the previous examples requirements have been defined graphically. Some users prefer, however, a pure textual definition because requirements can be formulated and inspected in a more compact form. It turned out that with current Modelica it is not possible to define requirements in a convenient way, if the requirement model contains a memory. For this reason, section 3.1 sketches a proposal for a Modelica extension to improve this situation.

3.1 Calling Blocks as Functions

The goal is to introduce functions with memory and events into Modelica. Since blocks already support memories and events, the simplest extension seems to be to introduce the feature that blocks can be called as functions. However, functions have a different type system than blocks: Arguments in functions can be identified by position, whereas in blocks they must be identified by name. For this reason, the “function calling” mechanism of a block is naturally restricted to named arguments. Since functions have an optional mechanism for named input arguments, but not for named output arguments, functions are generalized for named output arguments first. Afterwards, the optional calling mechanism of functions and the required calling mechanism of blocks are identical.

The basic idea is simple: (a) A block is called using its class name, (b) the inputs to the block call are defined by the usual modifiers of a block declaration, (c) one output of a block must be defined as return value of the call, by appending its name with “.name” to the “function call”. Take for example the block `MaxRising` of Figure 5. It could be expressed as a declaration in a pure textual form:

```
import Modelica_Requirements.ChecksInFixedWindow.*;
import Modelica_Requirements.Types.Property;

Property property = MaxRising(condition = start,
                              check = engineStart,
                              nRisingMax = 3).y;
```

Note, `(...).y` defines that output variable `y` of block `Modelica_Requirements.ChecksInFixedWindow.MaxRising` is computed and assigned to variable `property`. The above declaration is transformed (conceptually) to standard Modelica with a formal mapping rule resulting in:

```

MaxRising MaxRising_1(condition = start,
                       check = engineStart,
                       nRisingMax = 3);
Property property = MaxRising_1.y;

```

This shows that a tool has to introduce a declaration for an auxiliary component (here: `MaxRising_1`) and use the output of this block (here: `MaxRising_1.y`) in the expression where the call of `MaxRising` occurred.

The block calling can be nested in expressions. However, in order that the simple mapping rule above can be applied by a tool, several restrictions are necessary. Most importantly: A block can be called as a function *only* in the declaration section (with the additional restriction that it cannot be called in an if-expression). The proposed extension above was implemented in prototypes of Dymola and OpenModelica (Buffoni and Fritzson, 2014)

3.2 Examples

In the Modelica_Requirements library several textual examples are present in sub-library Examples.Textual, especially part of the EDF Backup Power Supply benchmark (Thuy, 2013). Example code:

```

Requirement R1(property=WhenRising(condition=Off,
                                   check=MPSVoltage < 170).y,
               text="MPS CAN be declared Off when
                   the voltage gets below 170 V");
Requirement R2(property=during(MPSVoltage < 160,
                                   check=Off),
               text="MPS MUST be declared Off when
                   the voltage gets below 160 V");

```

It is a matter of taste whether a user prefers a graphical or a textual definition – the Modelica_Requirements library supports both choices.

4 Utilizing Requirement Models

Once requirements are defined they are typically associated with behavioral models and various techniques are used to verify these requirements based on simulations. Integrating the modelled requirements manually in test scenarios of behavioral models may be a tedious task and there is a clear need to automate this process. Several proposals have been discussed within the MODRIO project for this purpose, especially (Bouskela et al., 2015; Schamai, 2013; Schamai et al., 2014) and also on using Modelica scripts for associating requirements with behavioral models. In (Elmqvist et al., 2015) two new Modelica language constructs are proposed to simplify this “automatic binding” task. These language elements are also useful for other applications, for example to compute the total mass of a multibody system or for contact handling.

The current development stage allows to check in every simulation run whether the defined requirements are satisfied or violated (or are not tested). Industrial applications would typically involve additional software on top of this base functionality, such as:

- *Monte Carlo Simulation*
Various initial conditions, operating points, and/or external disturbances are randomly generated within meaningful bounds and for every scenario simulations are performed. This brute force method for evaluation of dynamic systems is standard in many software tools.
- *TestWeaver*
TestWeaver (Junghanns et al., 2008) is a software tool from QTronic to construct automatically test scenarios, especially also for Modelica models. The goal of the tests is to drive the system in a state where it violates its specifications. A major application area are systems where the inputs have a countable number of values or areas (and these values vary over time).
- *Anti-Optimization*
A technique used at DLR-SR to evaluate controller designs, see e.g. (Joos, 2011): A special parameter optimization problem is formulated, in order to find an operating point of the system (e.g. height or speed of an aircraft), where the controller works as badly as possible. The major application area are systems where the operating region and the requirements are described by continuous signals.

5 Conclusions and Outlook

In this article the design of a new, open source Modelica library was presented to formally model requirements for industrial applications. The design was driven by applications of EDF (power plants, electrical systems) and Dassault Aviation (aircrafts). The basic design is based on the FORMAL Requirements Modeling Language FORM-L from (Thuy, 2014). The library in the current form (July 2015) is in an Alpha version. It is planned to additionally implement FORM-L components with overlapping sliding time windows, to include dynamic response and FFT requirement blocks from (Kuhn et al., 2015), to introduce continuous indicators for the properties where this is possible (in order that property blocks can be directly used as constraints or criteria for optimization-based methods), to add use cases of EDF and Dassault Aviation, and to connect the library to existing verification frameworks, such as TestWeaver.

Acknowledgements

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

Helpful discussions with Martin Kuhn (DLR) are appreciated. Furthermore, helpful discussions with members of the Modelica Association to define the fine details of the “calling-blocks-as-functions” approach, are also appreciated. Finally, improvement suggestions of the reviewers of this paper are appreciated.

References

- C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press. ISBN 978-0-262-02649-9, 2008.
- D. Bochvar. *Ob odnom trekhznachnom ischislenii i ego primenenii k analizu paradoksov klassicheskogo rasshirenogo funkcionnogo ischislenija*. In *Matematicheskij Sbornik* 4, no 46, pp. 287–308. 1937.
- D. Bouskela, N. Thuy, and A. Jardin. *D2.1.1 – Modelica extensions for properties modelling, Part II: Modeling Architecture for the Design Verification against System Requirements*. Internal report, ITEA2 MODRIO project, March 2015.
- M.A. Breuer. *A Note on Three-Valued Logic Simulation*. IEEE Transaction on Computer C-21, no. 4, pp. 399-402, 1972.
- L. Buffoni and P. Fritzson. *Expressing Requirements in Modelica*. Proceedings of the 55th International Conference on Simulation and Modeling (SIMS 2014), October 21-22, Aalborg, Denmark, 2014.
- Dassault Systèmes. *Dymola 2016.*, 2015. <http://www.Dymola.com>
- Department of Defense *Aircraft Electric Power Characteristics (MIL-STD-704F)*. 1984. Download: http://everyspec.com/MIL-STD/MIL-STD-0700-0799/MIL-STD-704F_1083/
- A. Garro, A. Tundis, and M. Otter. *D2.1.1 – Modelica extensions for properties modelling, Part IVb: FORM-L and Modelica: syntax and relationships*. Internal report, ITEA2 MODRIO project, Sept. 2014.
- H. Elmqvist, H. Olsson, and M. Otter. *Constructs for Meta Properties Modeling in Modelica*. Accepted for Modelica'2015 conference, 2015.
- A. Jardin and D. Bouskela. *D2.1.1 – Modelica extensions for properties modelling, Part I: Users motivation*. Internal report, ITEA2 MODRIO project, Sept. 2014.
- A. Jardin, D. Bouskel, N. Thuy, N. Ruel, E. Thomas, L. Chastanet, R. Schoenig, and S. Loembé. *Modelling of System Properties in a Modelica Framework*. Proceedings 8th Modelica Conference, Dresden, Germany, March 20-22., pp. 579-592, 2011. Download: <http://www.ep.liu.se/ecp/063/065/ecp11063065.pdf>
- H. D. Joos. *Worst-case parameter search based clearance using parallel nonlinear programming methods*. In: *Optimization based Clearance of Flight Control Laws*. Lecture notes in control and information sciences, 416. Springer, pp. 149-159, 2011. ISBN 978-3-642-22626-7. ISSN 0170-8643.
- A. Junghanns, J. Mauss, and M. Tatar. *TestWeaver – A Tool for Simulation-based Test of Mechatronic Designs*. Proceedings of the Modelica'2008 Conference, pp. 341-348, March 3-4, 2008. Download: <https://www.modelica.org/events/modelica2008/Proceedings/sessions/session3c4.pdf>
- M. Kuhn, M. Otter and T. Giese. *Model Based Specifications in Aircraft Systems Design*. Accepted for Modelica'2015 conference, Sept. 2015.
- L. Lamport. *Principles and Specifications of Concurrent Systems*, 2015. Hyberbook: <http://research.microsoft.com/en-us/um/people/lamport/tla/hyperbook.html>
- M. Leucker, and C. Schallhart, C. *A Brief Account of Runtime Verification*. Journal of Logic and Algebraic Programming 78, no. 5, pp. 293-303, 2009.
- J. Levy, S. Hassen, and T.E. Uribe. *Combining Monitors for Runtime System*. Electronic Notes in Theoretical Computer Science 70, no. 4, pp. 112-127, 2002.
- J. Łukasiewicz. *On three-valued logic*. In L. Borkowski (ed.), *Selected works by Jan Łukasiewicz*, North-Holland, Amsterdam, pp. 87–88, 1920. ISBN 0-7204-2252-3.
- Modelica Association. *Modelica, A Unified Object-Oriented Language for Systems Modeling. Language Specification, Version 3.3*, May 9, 2012. <https://www.modelica.org/documents/ModelicaSpec33.pdf>
- OMG. *Requirements Interchange Format (ReqIF)*, 2013. Download: <http://www.omg.org/spec/ReqIF/1.1/PDF/> <http://www.omg.org/spec/ReqIF/20110401/reqif.xsd>
- Open Source Modelica Consortium. *OpenModelica*, 2015. <https://openmodelica.org/>
- M. Otter M, L. Buffoni, P. Fritzson, M. Sjölund, W. Schamai, A. Garro, A. Tundis, and H. Elmqvist. *D2.1.1 – Modelica extensions for properties modelling, Part IV: Modelica for properties modeling*. Internal report, ITEA2 MODRIO project, Sept. 2014.
- N. Rescher. *Many-valued Logic*, McGraw-Hill, 1969.
- W. Schamai. *Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool*. Ph.D. Thesis, No. 1547, University of Linköping, 2013.. Download: <http://liu.diva-portal.org/smash/record.jsf?pid=diva2:654890>
- W. Schamai, L. Buffoni, and P. Fritzson. *An Approach to Automated Model Composition Illustrated in the Context of Design Verification*. Journal of Modeling, Identification and Control, volume 35- 2, pages 79–91, 2014.
- S. Steinhorst and L. Hedrich. *Targeting the Analog Verification Gap: State Space-based Formal Verification Approaches for Analog Circuits*. CAV 2009, Grenoble, France, 2009. Download http://www.em.cs.uni-frankfurt.de/FAC09/papers/FAC_09_Steinhorst.pdf
- N. Thuy. *D8.1.3 – Part 1 The Backup Power Supply*. Internal report, ITEA2 MODRIO project, Nov. 2013.
- N. Thuy. *D2.1.1 – Modelica extensions for properties modelling, Part III: FOrmal Requirements Modelling LAnguage (FORM-L)*. Internal report, ITEA2 MODRIO project, Sept. 2014.
- M. Tunnat. *Integration modellbasierter Methoden in den Entwicklungsprozess hybrider Flugzeugregelungssysteme am Beispiel des Ventilation-Control-System*. Master thesis, Technical University Hamburg-Harburg, Institut für Flugzeug-Kabinensysteme, 2011.

Towards a Formalized Modelica Subset[†]

Lucas Satabin¹ Jean-Louis Colaço¹ Olivier Andrieu¹ Bruno Pagano¹

¹Esterel Technologies/ANSYS SBU, France, firstname.lastname@ansys.com

Abstract

The ever growing requirement for safety in embedded systems, together with the willingness of having a modelling language to describe both the physics and the software that controls it makes Modelica an interesting candidate to design, simulate and implement complex systems. Originally designed to address multi-physics, since its version 3.3 Modelica integrates constructions to describe discrete controllers. Now the question of using Modelica to design critical embedded software arises.

In this paper we address the problem of defining a practical Modelica subset that can be entirely formalized and we sketch the formalization of this subset with the concrete example of static name resolution. This work should serve as a basis to define a suitable language that can be used to both simulate systems and generate embedded critical code.

Keywords: embedded systems, safety, code generation, formalization, name resolution

1 Introduction

Designing a complete programming language is a heavy task that involves many different aspects. The more features it contains, the more interactions between them are to be considered to ensure its correctness.

Modelica is an object-oriented language that was designed to simulate multi-physics systems. It is quite rich with a lot of constructs that are both static and dynamic. To make it a useful language, having a consistent behavior in its different implementations is a key point that can only be reached if it has a well documented and non-ambiguous semantics.

Moreover, the Modelica specification version 3.3 introduced new synchronous features that make it usable to design discrete controllers. It becomes tempting to use these features of Modelica to both simulate the physics with the controller and generate code for the controller, so that the same model is used for both activities.

Embedding code into critical systems (such as airplanes) requires some guarantees on the language and

tools used for their development; the most important ones are: *determinism* and *absence of ambiguities*. For example, implicit and undefined behaviors are problematic in such settings and would lead to additional verification activities (e.g. tests or reviews) to satisfy the certification objectives. Hence, the need for a programming language with unambiguous semantics appears clearly if one wants to use it in the development of safety-critical software.

Formalizing the language is a good way to ensure its correctness and analyze the safety issues it could raise.

In the scope of the CertMod project¹, we worked on formalizing the static semantics of a Modelica subset as a basis for a qualified code generator development. In the remainder of this paper we use the terms *qualification* and *certification* as defined in DO-330 (2011): “Tool qualification is the process necessary to obtain certification credit for a tool.” The current paper relates part of the results we produced during this project, which aims to provide a complete specification that can be used to develop a qualified code generator for Modelica. We focus on the basis elements identified in the scope of the CertMod project.

The contributions of this paper are the following:

- the definition of a practical subset of Modelica that can be formalized and used in a safety-critical context,
- a framework to formalize the various static aspects of Modelica and
- a formalization of static name resolution in Modelica.

The remainder of this paper is structured as follows: section 2 is a review of existing related work. Section 3 outlines the Modelica subset that is considered. Section 4 defines the formalization framework that will be used together with the notations. Section 5 depicts the name resolution within the formalism. Section 6 details the open points and future work.

[†]This work has been partially supported by the European Commission within the framework of the Clean Sky CertMod project with call identifier SP1-JTI-CS-2012-01.

¹http://cordis.europa.eu/project/rcn/111584_en.pdf

2 Related Work

Modelica Association (2012) is the reference document for Modelica specification; it describes in natural language in a pretty free style all the constructs of the language and their behaviours in different contexts. The description of a given construct can be scattered over the entire specification document. This makes it hard to ensure that an implementation entirely respects it. Formalizing the language requires to be systematic in the description, in the sense of identifying the different aspects of correctness (naming, typing, clocking, ...) and for each of them going through each construct and define its correction condition with respect to this aspect. One of the first benefits of a formalization is to provide an organization of the different concerns. It was already identified in Broman et al. (2006) that the Modelica specification could benefit from a more formal definition. The language has grown complex and a lot of constructs interact with each other, hence it has become hard to reason about Modelica models. Another benefit of a formal description is to reduce the possible interpretations to the intended one; this goal is reached by the use of mathematical and well defined notations.

Modelica was not designed with safety-critical embedded controllers in mind. This means that some language features are either not relevant or not defined appropriately for such applications. This was discussed in Thiele et al. (2012) where a Modelica sub- and superset was sketched to address safety requirements. Our subset is based on the one identified in this work, but we decided to define a strict subset and no superset of Modelica. This decision to have a strict subset is motivated by the willingness to seamlessly integrate with existing implementations. No change is required between the model being simulated and the one generating the actual code.

Implementations compliance rapidly arises when several implementations of Modelica exist and different behaviors are observed. For example, **protected** elements in OpenModelica² may be accessed with the dot-notation whereas Dymola³ does not allow to access **protected** classes. Also, the specification may be incomplete on some points and implementations must interpret it. For instance, defining the scope in which re-declaration as modifications takes place is subject to controversy⁴.

```
class A
  replaceable class R end R;
end A;

class S type T = Real; end S;
```

²<https://openmodelica.org/>

³<http://www.3ds.com/products-services/catia/products/dymola>

⁴For instance <https://trac.modelica.org/Modelica/ticket/1680>

```
class B
  extends A(redeclare class R = S);
  extends C;
end B;

class C
  class S type T = Integer; end S;
end C;
```

According to the specification, it is unclear what $B.R.T$ represents, whether it is $C.S.T$ (i.e. `Integer`) or $.S.T$ (i.e. `Real`). Even though the various implementations agree on this particular ambiguity, it is still problematic in the context of a certification process, because the specification is the reference, not the implementation.

To address such problems, test suites can help disambiguating situations. A Modelica compliance test suite⁵ is being developed that aims to validate various implementations of Modelica. Such test suites are useful to validate a compiler but can become hard to maintain up-to-date over time. In any case, these tests need oracles to validate implementation outputs and these oracles must be defined by the language specification. This is particularly important in a certification process such as DO-178C (2011), which requires to have test oracles based upon the specification. The typical approach used for software development is to write requirement based specification documents and test oracles are written using these requirements. This process allows for a clear traceability between requirements and test cases.

As of today, if the goal were to implement a qualified code generator for Modelica, the specification from Modelica Association (2012) coupled with a test suite would probably not satisfy a certification authority requirements.

In the industry, languages used to write embedded controllers are not all formalized. For instance, the C programming language is standardized⁶ but implementations of certain constructs diverge depending on the compiler or the target platform. In the embedded software world, some rules and constraints are widely accepted and used to define a subset of C that aims to be safer. These guidelines are known under the name MISRA C⁷.

The ultimate step in the formalization direction is having a formally described language and formally proven compiler, which gives a comprehensive formal proof that each transformation in the compiler preserves the semantics of the input program. The most advanced work in this area is incarnated by the CompCert C compiler Leroy (2009).

Finally, in the model-based approach to embedded software development, Scade 6 is the industrial dialect of the dataflow language Lustre, Halbwachs et al. (1991), extended with state machines, Colaço et al. (2005).

Since the latest major evolution of the language called

⁵<https://github.com/modelica-compliance>

⁶For example by the ISO/IEC 9899:1999 aka. C99 standard.

⁷<http://www.misra.org.uk/>

Scade 6, the entire language static semantics is formally defined by various type systems that cover all constructs of the language. This work follows the approach chosen in the design of Lucyd Synchrone Pouzet (2006). This formalization is the basis of the Scade 6 certified compiler implementation. The present work is based on the very same idea and aims to provide a similar formalism level for a Modelica subset.

3 A Practical Modelica Subset

As we mentioned in the introduction, Modelica has a lot of constructs. Historically designed to model multi-physics systems with continuous time, it gains only recently the ability to describe synchronous controllers. In the scope of qualified embedded controllers development, only these synchronous features are of interest. Moreover, continuous time features are hard to formally describe and lots of behaviors depend on the solver at runtime. That is why we made the choice to formally describe a Modelica subset instead of taking the complete language.

To be of any practical use, the subset must be as complete as possible so that its expressiveness is not sacrificed for the sake of the formalization simplicity.

A first subset was described in Thiele et al. (2012), which was quite conservative. For example, import clauses were excluded from this subset. This kind of restriction can rapidly become annoying when dealing with existing libraries that make heavy use of import clauses (including the Modelica standard library). Our work is based on this subset with some additions to make it a more realistic subset.

3.1 Declarations

In the Modelica specification the language is defined as an *EBNF*⁸ but syntactically allows for incorrect constructs. For instance, the EBNF does not prevent one from writing

```
function F
  input Integer i;
  output Integer o;
  equation
    o = i * 3;
end F;
```

This kind of class declaration is illegal (Modelica Association, 2012, section 12.2) as functions may not have equations but only statements in an **algorithm** section. However, this declaration is syntactically allowed (Modelica Association, 2012, section 4.5).

In comparison, the subset aims to syntactically enforce as many constraints as possible. Syntactically enforcing constraints allows for less normalization steps and checks after parsing, and makes the formalism simpler.

⁸Extended Backus-Naur Form

We added more syntactical restrictions on declarations. For example a package can only be defined inside a package and not inside other specialized classes. The same constraint exists for functions, which can only be declared in packages. Having a function declared in something else than a package makes it parameterized by all components of the class it is declared in. This restriction was thought as a way to improve modularity. In the following, we do not reason about the flattened model but about the structured input models. Checks that are described can be done in a modular way (i.e. without effective computation of the flattened model). Modifications in classes makes modular reasoning more complex. Also, we see functions as pure functions (Modelica Association, 2012, section 12.3) in the sense that they are side-effect free, and thus must not depend on the context of instantiation.

Modelica also allows many of type prefixes, or modifiers, for components. They are not all present in the subset. For instance, **inner** and **outer** components are not included, as they introduce an implicit binding that makes it hard to reason about. We will discuss this in section 6.

Declarations in Modelica can also be redeclared in inheriting classes. This feature makes it possible to change the behavior of an inherited component by replacing it with another component. Although it is not forbidden by certification processes such as DO-178C (2011) and its object-oriented extension DO-332 (2011), we identify this feature as dangerous. Replacing or redeclaring components in this context requires more checks and validation to be performed to ensure that the global behavior and invariant of the inherited model are respected. This feature is not included in the subset, however parameter modifications are.

3.2 Equations

The selected subset contains basically all kind of equations that are meaningful in the context of synchronous models. It means the subset accepts these equations:

- simple equations that are flow definitions.
- **if**-equations
- clocked **when**-clauses
- **connect**-clauses

The only missing equation kind are **for**-equations. Their general form as defined in (Modelica Association, 2012, section 8.3.2) may introduce patterns that cannot be statically verified. Even though the expression the loop iterates over is required to be a parameter, it still allows to multiply define some cells of a vector or to leave some other cells non-initialized. However, adding **for**-equations that reduce to a *map* operator will be considered in future developments.

3.3 Expressions

The restrictions on the expressions are essentially the same as in Thiele et al. (2012). All continuous-time related operators are not included as they do not make sense in this context. On the other hand, most of the expressions related to synchronous features are included.

The aim of this paper is not to describe the subset entirely, and the complete grammar could not fit here. For the complete, refer to Satabin et al. (2015).

4 Formalization framework

The main contribution of this paper is to define a framework that can be used to formalize various aspects of the Modelica language. In programming language theory, it is used to distinguish two aspects of a language semantics:

- the *static semantics*, which corresponds to a certain (language dependent) level of correctness of syntactically correct programs required before execution, this aspect is statically checked at compile-time (i.e. without execution) and
- the *dynamic semantics*, which describes the behavior of the programs that are both syntactically and statically correct.

This separation reduces the set of programs to be considered by the dynamic semantics, in which one can assume that all the static aspects are respected.

In this work we focused on the static semantics of Modelica, which encloses:

- Static name lookup (Modelica Association, 2012, section 5.3).
- Type checking (Modelica Association, 2012, chapters 6 to 14).
- Clock checking (Modelica Association, 2012, chapter 16).

For each of these aspects we defined a dedicated system of inference rules, derived from the Modelica specification. The formalism used is based on works such as Igarashi et al. (2001), Igarashi (1999) for the object-oriented and type part or Forget et al. (2008) for the clock checking.

The Modelica syntax is rich and each construct may have several shapes. While writing a formalization it is more readable to have only one shape for each construct. That is why, the first step before formalizing is the normalization of declarations.⁹

⁹Note that this normalization must preserve correctness i.e. an incorrect program cannot normalize into a correct one and reciprocally.

4.1 Component Clauses

Components in Modelica are declared with component clauses. One such clause can declare several components of different array types. Moreover, clauses are grouped into public and protected sections which defines the visibility of each component declared in this section. Even though these syntactic constructs are allowed in our subset, component clauses are normalized so that:

- each clause declares exactly one component;
- each clause has a visibility, written ν , which corresponds to the section it is declared in;
- each clause has a set of modifiers (with restrictions discussed in section 3) written μ . If a declaration has no modifiers μ is the empty set, written \emptyset ;
- each array subscript appears after the component name.

The normalization of component clauses is depicted in figure 1 where :

- c, c_1, \dots, c_i represent component declarations with potential array subscripts ;
- T represents a type identifier with potential array subscripts and
- t is a type identifier.

Hence, a component declaration is written $\nu \mu T c$.

We will also use lists of components in the following which will be written $\overline{\nu \mu T c}$. This notation is a shortcut for $\nu_1 \mu_1 T_1 c_1, \dots, \nu_n \mu_n T_n c_n$ for some $n \in \mathbb{N}^*$

4.2 Short Class Definitions

Modelica allows for so-called short class definition (Modelica Association, 2012, section 4.5.1). It is presented as syntactic sugar for simplified standard class definitions which does not introduce a new scope. Our subset allows for such declarations only for **type** and **connector**. The normalizing function *rewriteShort* is given in figure 2.

The component name λ is a fresh name which is generated during rewriting. Referring to a component whose type is declared with short class definition is equivalent to accessing the λ component. Enumerations are not rewritten because their only possible shape is with the short class definition. In the following, special rules will be written to handle them.

$$\begin{aligned} \text{normDecl}(\nu \mu \text{ t } [n_1, \dots, n_p] \text{ c};) &= \nu \mu \text{ t } \text{ c}[n_1, \dots, n_p]; \\ \text{normDecl}(\nu \mu \text{ T } \text{ c}_1, \dots, \text{ c}_q;) &= \text{normDecl}(\nu \mu \text{ T } \text{ c}_1;) \dots \text{normDecl}(\nu \mu \text{ T } \text{ c}_q;) \\ \text{normDecl}(\nu \mu \text{ t } \text{ c};) &= \nu \mu \text{ t } \text{ c}; \end{aligned}$$

Figure 1. Normalization of component clauses

$$\begin{aligned} \text{rewriteShort}(\text{connector } \text{ C } = \mu \text{ T}) &= \text{connector } \text{ C } \mu \text{ T } \lambda; \text{ end } \text{ C} \\ \text{rewriteShort}(\text{type } \text{ C } = \mu \text{ T}) &= \text{type } \text{ C } \mu \text{ T } \lambda; \text{ end } \text{ C} \\ \text{rewriteShort}(\text{type } \text{ C } = \mu \text{ T}[n]) &= \text{type } \text{ C } \mu \text{ T } \lambda[n]; \text{ end } \text{ C} \end{aligned}$$

where each occurrence of λ is fresh.

Figure 2. Normalization of short class definitions

4.3 Names

Components in Modelica are referred to by either simple names of the form C or by composite names of the form $A.B.C$ (Modelica Association, 2012, chapter 5). These composite names, or paths, can be absolute, in which case they start with an dot, as in $.A.B.C$. To handle all paths uniformly in the upcoming formalization, we introduce the root package name, written \star . Absolute paths are thus written $\star.A.B.C$ and all paths have the same shape.

In the following, we will differentiate between absolute resolved paths and unresolved paths (which can be either relative or absolute). For the sake of readability we will use notation \underline{P} for absolute paths of the form $\star.A.B.C$ and \underline{p} for unresolved paths of the form $A.B.C$.

4.4 Class Table

A Modelica model usually contains several classes organized into packages. These classes are stored in a table, written CT , that maps absolute class paths to their definition. A same path can only refer to at most one class definition. Construction of CT is done by walking through the syntactic structure of the model and by adding each encountered class definition name prefixed by its enclosing package path. This construction may fail if two classes are located at the same path. If it succeeds, all classes of the model are present in this table. The function dom is used to check whether an absolute path is an existing class with the notation $A.B.C \in dom(CT)$.

In the remainder of this paper, we consider that CT was successfully built.

After the short class definition rewriting that was discussed previously we can see classes in CT as the sets of components that are syntactically declared in them. For example, let's consider the class C below:

```
class C;
  Integer C1;
  parameter Boolean C2;
end C;
```

Conceptually this class is equivalent to the set of component declarations $C1$ and $C2$, written $\{\text{Integer } C1; \text{parameter Boolean } C2\}$. We will use the notation $\text{Integer } C1 \in CT(A.B.C)$ as a way to express the fact that a component is declared in a class.

4.5 Specialized Classes

The Modelica specification defines several specialized classes (Modelica Association, 2012, section 4.6). Most of the time, the specialized class kind does not matter, and they all are treated the same way and we will use the notation \mathbf{ckind} to denote any specialized class kind. However sometimes the kind of specialized class is relevant to check some restrictions or allow some extensions. To this end, we define a function named $kind$, depicted in figure 3, that, given a class absolute path, returns the kind of specialized class it represents.

$$\begin{aligned} \text{kind}(\star) &= \text{package} \\ \text{kind}(\underline{C}) &= \mathbf{ckind} \text{ if } CT(\underline{C}) = \nu \mathbf{ckind } \text{ C} \dots \text{end } \text{ C} \\ \text{kind}(\underline{C}) &= \text{type} \text{ if } CT(\underline{C}) = \nu \text{ type} = \text{enumeration}(\dots) \end{aligned}$$

Figure 3. Specialized class kind

5 Name Resolution

As part of the formalization of Modelica's static semantics, the first aspect to consider is the name resolution. It is crucial in the sense that there must exist no ambiguity on what is referred to when a name is used in a model and neither correction can be decided nor compilation

done without linking referenced names to the definition of the identified entity. Modelica has several features that are involved in this step and several rules that must be respected. It has modularization features, such as packages and visibility, that are to be taken into account.

In this section we propose a formalism for name resolution in our subset discussed in 3. It is written as a bunch of inference rules, each of which will be linked to the sections in the Modelica specification it was derived from.

5.1 Import Clauses

Classes and components can be imported in other classes to shorten the name that are referred to. There are four kinds of import clauses in Modelica (Modelica Association, 2012, section 13.2.1):

1. **import** A.B.C where C becomes visible in the lexical scope of the **import** clause.
2. **import** A.B.{C, D, E} where C, D and E become visible in the lexical scope of the **import** clause
3. **import** A.B.* where all elements defined in A.B become visible in the lexical scope of the **import** clause.
4. **import** D = A.B.C where A.B.C becomes visible with name D in the scope of the **import** clause.

Clauses of the second form can be reduced to case one by duplicating **import** clauses as many times as there are imported elements and will be treated as such in the following. In case three the import clause is said to be *unqualified* and has lower priority than other import clauses that are said to be *qualified* (Modelica Association, 2012, section 5.3.1). Case four allows to introduce a different local name for imported elements that otherwise would conflict.

Imported names are always fully qualified names (Modelica Association, 2012, section 13.2.1.1). It means that if one writes **import** A in Modelica, it will be treated as **import** *.A. In other words, only absolute composite names are imported.

We define the *imports* function that will be used in the following to get the list of unresolved **import** from a resolved path. Each import returned by this function is the pair containing the import name and the imported path. In the case of unqualified imports, the empty set symbol \emptyset is returned instead of a name.

$$\text{imports}(\underline{C}) = \{ \text{namePath}(\text{imp}) \mid \text{imp} \in CT(\underline{C}) \}$$

where function *namePath* is defined by:

$$\begin{aligned} \text{namePath}(\text{import } \underline{A}.B) &= (B, \underline{A}.B) \\ \text{namePath}(\text{import } C = \underline{A}.B) &= (C, \underline{A}.B) \\ \text{namePath}(\text{import } \underline{A}.*) &= (\emptyset, \underline{A}) \end{aligned}$$

5.2 Inheritance

Modelica is an object-oriented language that allows for multiple inheritance (Modelica Association, 2012, section 7.1.1). A class may contain as many **extends** clauses as wanted in any order. We define the function *extends* which returns the list of unresolved extended path of a resolved path.

$$\text{extends}(\underline{C}) = \{ \underline{X} \mid \text{extends } \underline{X} \in CT(\underline{C}) \}$$

5.3 Visibility

Modelica defines two level of visibility: **public** and **protected**. The **protected** visibility means that the element cannot be accessed via the dot notation (Modelica Association, 2012, section 4.1). Visibility appears in several kinds of clauses: **extends** clauses, component clauses and class definition. An **extends** clause may be protected, which means that all inherited components and classes are considered **protected** from the inheriting class (Modelica Association, 2012, section 7.1.2).

When resolving names, we would need to check that a name is visible when accessing it with the dot notation (Modelica Association, 2012, section 4.1).

5.4 Static Name Lookup

Based on the previous definitions, we can define name lookup in our Modelica subset. It starts with the static name lookup, where all the classes and their component names are resolved. The complete set of rules are depicted in figure 4.

Judgements of these rules must be read as follows:

- $\underline{P} \vdash C \overset{\bullet}{\Rightarrow} \underline{D}$ means “the simple name C seen from \underline{P} is resolved to path \underline{D} .”
- $\underline{P} \vdash C \overset{\circ}{\Rightarrow} \underline{D}$ means “the simple name C seen from \underline{P} is resolved to path \underline{D} by only using named elements of \underline{P} or its super classes.”
- $\underline{P} \vdash C \overset{\circ}{\Uparrow}$ means “the simple name C seen from \underline{P} cannot be resolved by only using named elements of \underline{P} or its super classes.”
- $\underline{P} \vdash C \overset{\bullet}{\Uparrow}$ means “the simple name C seen from \underline{P} cannot be resolved by using named elements of \underline{P} or its super classes nor import clauses of \underline{P} .”

Conceptually, the class path on the left of the \vdash symbols gives the scope of the lookup and unambiguously describes where the search must start.

Several aspects of the static name lookup are of interest in this formalization. First, visibility is not taken into account. The reason why and impacts will be discussed in section 6. Then, we can see that few rules are needed

$$\begin{array}{c}
\text{N-Root} \frac{}{\vdash \star \overset{\circ}{\Rightarrow} \star} \quad \text{N-SELF} \frac{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}} \overset{\circ}{\Rightarrow} \underline{\mathbf{D}}}{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{D}}} \quad \text{N-INCT} \frac{\underline{\mathbf{P}}.\underline{\mathbf{C}} \in \text{dom}(CT) \quad \forall \mu \underline{\mathbf{T}} \underline{\mathbf{C}} \notin CT(\underline{\mathbf{P}})}{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}} \overset{\circ}{\Rightarrow} \underline{\mathbf{P}}.\underline{\mathbf{C}}} \\
\text{N-COMP} \frac{\underline{\mathbf{P}}.\underline{\mathbf{C}} \notin \text{dom}(CT) \quad \forall \mu \underline{\mathbf{T}} \underline{\mathbf{C}} \in CT(\underline{\mathbf{P}})}{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}} \overset{\circ}{\Rightarrow} \underline{\mathbf{P}}.\underline{\mathbf{C}}} \\
\text{N-SUPER} \frac{\underline{\mathbf{C}}.\underline{\mathbf{D}} \notin \text{dom}(CT) \quad \forall \mu \underline{\mathbf{X}} \underline{\mathbf{D}} \notin CT(\underline{\mathbf{P}}) \quad \underline{\mathbf{X}} \in \text{extends}(\underline{\mathbf{C}}), (\underline{\mathbf{C}} \vdash \underline{\mathbf{X}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{Y}} \wedge \underline{\mathbf{Y}} \vdash \underline{\mathbf{D}} \overset{\circ}{\Rightarrow} \underline{\mathbf{T}}) \quad \forall \underline{\mathbf{Z}} \in \text{extends}(\underline{\mathbf{C}}), \underline{\mathbf{C}} \vdash \underline{\mathbf{Z}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{W}} \wedge (\underline{\mathbf{W}} \vdash \underline{\mathbf{D}} \overset{\circ}{\Rightarrow} \underline{\mathbf{T}} \vee \underline{\mathbf{W}} \vdash \underline{\mathbf{D}} \overset{\circ}{\Uparrow})}{\underline{\mathbf{C}} \vdash \underline{\mathbf{D}} \overset{\circ}{\Rightarrow} \underline{\mathbf{T}}} \\
\text{N-NOSELF} \frac{\underline{\mathbf{P}}.\underline{\mathbf{C}} \notin \text{dom}(CT) \quad \forall \mu \underline{\mathbf{X}} \underline{\mathbf{C}} \notin CT(\underline{\mathbf{P}}) \quad \forall \underline{\mathbf{X}} \in \text{extends}(\underline{\mathbf{P}}), \underline{\mathbf{P}} \vdash \underline{\mathbf{X}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{Y}} \wedge \underline{\mathbf{Y}} \vdash \underline{\mathbf{C}} \overset{\circ}{\Uparrow}}{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}} \overset{\circ}{\Uparrow}} \\
\text{N-IMPORTQUAL} \frac{(D, \underline{\mathbf{X}}.\underline{\mathbf{Y}}) \in \text{imports}(\underline{\mathbf{C}}) \quad \star \vdash \underline{\mathbf{X}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{P}} \quad \underline{\mathbf{P}} \vdash \underline{\mathbf{Y}} \overset{\circ}{\Rightarrow} \underline{\mathbf{E}} \quad \text{kind}(\underline{\mathbf{P}}) = \text{package}}{\underline{\mathbf{C}} \vdash \underline{\mathbf{D}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{E}}} \\
\text{N-IMPORTUNQUAL} \frac{\underline{\mathbf{C}} \vdash \underline{\mathbf{D}} \overset{\circ}{\Uparrow} \quad (D, \underline{\quad}) \notin \text{imports}(\underline{\mathbf{C}}) \quad (\emptyset, \underline{\mathbf{X}}) \in \text{imports}(\underline{\mathbf{C}}) \quad \star \vdash \underline{\mathbf{X}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{P}} \quad \underline{\mathbf{P}} \vdash \underline{\mathbf{D}} \overset{\circ}{\Rightarrow} \underline{\mathbf{E}} \quad \text{kind}(\underline{\mathbf{P}}) = \text{package}}{\underline{\mathbf{C}} \vdash \underline{\mathbf{D}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{E}}} \\
\text{N-NOIMPORT} \frac{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}} \overset{\circ}{\Uparrow} \quad (D, \underline{\quad}) \notin \text{imports}(\underline{\mathbf{C}}) \quad \forall (\emptyset, \underline{\mathbf{X}}) \in \text{imports}(\underline{\mathbf{P}}), \underline{\mathbf{P}} \vdash \underline{\mathbf{X}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{Y}} \wedge \underline{\mathbf{Y}} \vdash \underline{\mathbf{C}} \overset{\circ}{\Uparrow}}{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}} \overset{\bullet}{\Uparrow}} \\
\text{N-ENCL} \frac{CT(\underline{\mathbf{P}}) = \text{ckind } \underline{\mathbf{P}} \dots \text{end } \underline{\mathbf{P}} \quad \underline{\mathbf{P}}.\underline{\mathbf{C}} \vdash \underline{\mathbf{D}} \overset{\circ}{\Uparrow} \quad \underline{\mathbf{P}} \vdash \underline{\mathbf{D}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{E}}}{\underline{\mathbf{P}}.\underline{\mathbf{C}} \vdash \underline{\mathbf{D}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{E}}} \\
\text{N-ENCAPS} \frac{CT(\underline{\mathbf{P}}) = \text{encapsulated ckind } \underline{\mathbf{P}} \dots \text{end } \underline{\mathbf{P}} \quad \underline{\mathbf{P}}.\underline{\mathbf{C}} \vdash \underline{\mathbf{D}} \overset{\circ}{\Uparrow} \quad \star \vdash \underline{\mathbf{C}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{E}}}{\underline{\mathbf{P}}.\underline{\mathbf{C}} \vdash \underline{\mathbf{D}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{E}}} \\
\text{N-DOT} \frac{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{D}} \quad \underline{\mathbf{D}} \in \text{dom}(CT) \quad \underline{\mathbf{D}} \vdash \underline{\mathbf{E}} \overset{\circ}{\Rightarrow} \underline{\mathbf{F}} \quad \text{kind}(\underline{\mathbf{D}}) = \text{package}}{\underline{\mathbf{P}} \vdash \underline{\mathbf{C}}.\underline{\mathbf{E}} \overset{\bullet}{\Rightarrow} \underline{\mathbf{F}}}
\end{array}$$

Figure 4. Static Name Lookup

to formally describe the entire static lookup semantics. It represents aspects that are scattered in the specification, all put together here in a unified framework.

N-ROOT indicates that the root package name \star always resolves to itself. It is the base case when looking up in the enclosing classes. All predefined types (such as `Integer`) and predefined functions (such as `abs`) are considered to be defined in the root package.

N-SELF simply states that if a simple name can be resolved with the named elements of a class or its super classes, then it is resolved. It is kind of a weakening rule for name resolution since the premise gives a stronger information than the conclusion.

The base rules N-INCT and N-COMP look up for a simple name in the current class. They state that if a simple name is declared in the current class, then it resolves to this name augmented with the path of the current class. The same name cannot be defined twice in the same class (Modelica Association, 2012, section 5.6.3).

The rule N-SUPER treats the case where a simple name is defined in inherited classes. A same name C can be inherited multiple times if and only if all inherited elements with name C are exactly identical (Modelica Association, 2012, section 7.1). In this rule *identical* means that the resolved path is the same for all inherited elements with name C .

If the rules we saw so far do not apply to resolve a simple name C , then we conclude that it is not defined in the current class. This is what the rule N-NOSELF means.

In such a case, the rules N-IMPORTQUAL and N-IMPORTUNQUAL may be applied, to lookup for the simple name in the import clauses. The former rule looks up for the name in qualified imports, while the latter one looks up in unqualified import if no qualified import clause allowed to resolve the name (Modelica Association, 2012, section 5.3.1). Imported paths are resolved starting in the root package (Modelica Association, 2012, section 13.2.1.1) and names can only be imported from packages (Modelica Association, 2012, section 13.2.1.2). Our subset allows packages to be defined only packages, that is why it is sufficient to check that only the last element of the path is a package.

If none of the import-related rules described in the previous paragraph applies to resolve a simple name C , then we conclude that it is not defined in the current class, nor is it imported. This is the meaning of rule N-NOIMPORT.

Only in this case, the simple name must be resolved by looking up in the enclosing classes. Two different cases may apply at this point depending on the definition of the current class. If the current class is declared **encapsulated**, rule N-ENCAPS applies and the name is looked up in the root package (Modelica Association, 2012, section 5.3.1). In the case the class is not **encapsulated**, rule N-ENCL applies and the name is looked up in the directly enclosing class.

Finally, the last case deals with composite names. The set of rule deals only with static name resolution, which

means that composite names corresponding to component accesses of class instances are not treated here. We will discuss such cases in section 5.5. Static resolution of composite names is only allowed for names defined in packages, as stated by rule N-DOT. The last name in the path is looked up among elements defined or inherited in the package resolved so far (Modelica Association, 2012, section 5.3.2).

5.5 Component Lookup

In the previous section we covered the static name lookup only. This is, the resolution of class names in packages and component names in packages. Composite names that access components inside components require some typing information to be resolved. They indeed require to be aware of the structure of the component to decide what component the name represents. This structure is only known once all static names are resolved. We can then gather the list of components in a component using the *components* function depicted in figure 5. The *extendComponents* function allows to retrieve all inherited components.

Components of a resolved class are all the components defined in this class or inherited. Resolving accesses to components is done by the type checking. The type system is beyond the scope of this paper, but the typing rule T-DOT which describes component access in a component would look like this.

$$\text{T-DOT} \frac{n : \underline{\mathbf{T}} \quad \nu \mu \underline{\mathbf{C}} \quad c \in \text{components}(\underline{\mathbf{T}})}{n.c : \underline{\mathbf{C}}}$$

It reads as: if a simple name n has a resolved type $\underline{\mathbf{T}}$, then we can resolve and type $n.c$ if c is a component of $\underline{\mathbf{T}}$ with type $\underline{\mathbf{C}}$. Of course this rule is just a sketch and more concepts are taken into account by the real type system.

5.6 Class Resolution

A class in Modelica is said to be resolved if several constraints are respected:

- All component types can be resolved ;
- All **import**-clauses can be resolved ;
- All **extends**-clauses can be resolved ;
- All components defined in a class must have names distinct from inherited components. In Modelica component may have the same name if they are syntactically equal (Modelica Association, 2012, section 7.1). The specification recommends to emit a warning in this case, but we decided to forbid it, as it does not bring anything to define twice components that are exactly the same, and most probably it is a symptom of model design problem ;

$$\begin{aligned}
 \text{extendComponents}(\underline{\mathbf{C}}) &= \bigcup_{\underline{\mathbf{D}} \in \text{extends}(\underline{\mathbf{C}})} \{ \nu_x \mu_x \underline{\mathbf{X}} \ x \mid \nu_x \mu_x \underline{\mathbf{X}} \ x \in \text{components}(\underline{\mathbf{D}}) \} \\
 \text{components}(\underline{\mathbf{C}}) &= \begin{cases} \{ \text{public } \underline{\mathbf{C}} \ E_i \mid i \in [1..n] \} & \text{if } CT(\underline{\mathbf{C}}) = \text{enumeration}(E_1, \dots, E_n) \\ \{ \nu \mu \underline{\mathbf{X}} \ x \mid \nu \mu \underline{\mathbf{X}} \ x \in CT(\underline{\mathbf{C}}) \} \cup \text{extendComponents}(\underline{\mathbf{C}}) & \text{otherwise.} \end{cases}
 \end{aligned}$$

Figure 5. Component Lookup Function

- All qualified **import**-clauses to distinct names ;
- All unqualified **import**-clauses bring distinct names into scope ;

Rule N-CLASS in figure 6 gives the rule that ensures that all names are resolved in a class. And that all constraints defined above are respected.

6 Discussion and Future Work

In the context of the CertMod project, we also formalized type checking and clock checking of models based on similar rules. This work is the basis that we used to write a complete Modelica front-end that performs all static checks we described on input models. It can also be used to write or verify oracles in a compliance test suite, and then test the model checker against these oracles.

Some restrictions present in the current subset could be removed, and some omissions could be added. For instance, we did not take visibility of elements into account. This was motivated by our tests on various Modelica implementations which did not agree with neither the specification nor between each other. Adding visibility to the name resolution rules would be quite easy though. Only rules N-IMPORT and N-DOT would need to take this visibility into account. The *extends* function would also require to return the visibility of the extends clause. Similarly the sketched T-DOT rule would require that ν is **public**.

Other constructs that were not taken into account in name resolution rules in this paper are **inner/outer** declarations (Modelica Association, 2012, section 5.4). These constructs introduce an implicit name, inherited from an enclosing class. They represent a handy way of having global parameters in a model that we do not bother passing explicitly to each part requiring it. Adding these constructs to the subset would require to add rules to resolve **outer** names. These rules would be quite complex, considering the restrictions and constraints that exists on them. The rules must represent the fact that the closest **inner** component with the same name is selected when the class is instantiated.

Redeclarations in inheriting classes are also not included in our subset. Redeclaring classes allows for having a class name denoting a completely different path in a sub-class than in the inherited one. Remember the lookup scope problem for redeclarations we discussed in section 2. Formalizing redeclarations would definitively

help clarify the situation by having a non ambiguous way of describing the lookup scope. However, the resulting rules would be quite complex because for each name one should lookup for the current redeclaration, if any.

This added complexity makes it harder to read and understand the rule, but is symptomatic of an intrinsic complexity in the language construct. As a rule of thumb, the fact that a construct introduces complexity in the formalism can be seen as a hint whether the construct is legit or not. A too high complexity reflects a construct that will be hard to understand for modelers, and to implement correctly by tool providers.

7 Conclusion

In this work, we described all the rules related to name resolution as described in the Modelica specification. It was interesting and enlightening to compile the rules and constraints that appear at various places in the specification into a single place. It also allowed us to detect some features that may be problematic to write a qualified code generator for Modelica. For example, in the rules depicted in figure 4, the involved concepts are usual in object-oriented languages. However, the unqualified import clause lookup described by rule N-IMPORTUNQUAL implies a priority in name lookup that would require more validation activities to be used. The mix with qualified imports makes it also harder for the modeler to determine which element is selected. The safest way to deal with this problem would be to avoid unqualified imports all together, and to exclude them from the subset. Moreover, the encapsulated concept appears to be quite exotic and would also require extra checks to be performed as it introduces some irregularities in the lookup algorithm. Language features must ensure the highest possible level of safety, and restricting some constructs can benefit to developers. Expressiveness is important in a language but for safety-critical software development, safety and non ambiguity is even more important.

The considered subset presented here only includes discrete synchronous features of Modelica and the formalization only deals with static aspects of this subset. Adding the dynamic semantics of the subset appears to be an important step to take to achieve a comprehensive formal description of the language. Such a semantics would describe how a model behaves when it is instantiated and how the generated code must behave as well. This can be used to write oracles in the test suite and then

$$\begin{array}{l}
\forall \nu \mu \underline{x} \underline{x} \in CT(\underline{C}), \underline{C} \vdash \underline{x} \dot{\Rightarrow} _ \\
\forall (_, \underline{x}) \in imports(\underline{C}), \underline{C} \vdash \underline{x} \dot{\Rightarrow} _ \\
\forall \underline{x} \in extends(\underline{C}), \left(\underline{C} \vdash \underline{x} \dot{\Rightarrow} \underline{x} \wedge \forall \nu_c \mu_c C \underline{C} \in CT(\underline{C}), \underline{x} \vdash C \overset{\circ}{\uparrow} \right) \\
\forall ((N, \underline{x}), (N, \underline{y})) \in imports(\underline{C}) \times imports(\underline{C}), \left(\star \vdash \underline{x} \dot{\Rightarrow} \underline{D} \wedge \star \vdash \underline{y} \dot{\Rightarrow} \underline{D} \right) \\
\forall ((\emptyset, \underline{x}), (\emptyset, \underline{y})) \in imports(\underline{C}) \times imports(\underline{C}), \left(\begin{array}{l} \star \vdash \underline{x} \dot{\Rightarrow} \underline{A} \\ \wedge \star \vdash \underline{y} \dot{\Rightarrow} \underline{B} \\ \wedge \underline{A} \vdash C \overset{\circ}{\Rightarrow} _ \implies \underline{B} \vdash C \overset{\circ}{\uparrow} \end{array} \right)
\end{array}$$

\underline{C}

Figure 6. Class Resolution

validate simulators as well as code generators and check that they agree on the behavior through the test suite.

A complete formal semantics of a language brings also the possibility to write proofs on the language. This is useful to ensure that the type-system is sound and that the language has a deterministic behavior. Reaching this point naturally requires a lot more work to be done, and the continuous part of Modelica would be quite problematic to semantically describe.

The presented work is a first small step toward having a formally described version of Modelica. Although we only covered a small part of the various aspects of the language, it sets up a framework for a more comprehensive formalization. It already brings some clarity where rules written in English may be misinterpreted. It is also a comprehensive, concise and non-ambiguous way to describe these rules. We believe that it is a huge step forward and that it can help clarifying things when it is hard to interpret the specification. We also believe that this work can help in writing the next versions of the Modelica specification. Not necessarily does it mean that this exact formalism must be included in it, but having this way of describing behaviors in mind helps writing more comprehensive and rigorous specification.

Acknowledgments

We would like to particularly thank Martin Otter, Bernhard Thiele and Daniel Schlabe for their precious help and their answers during this work. We also want to thank Marc Pouzet for his comments on the clock calculus we developed in this project.

References

David Broman, Peter Fritzon, and Sébastien Furic. Types in the modelica language. In *Proceedings of the Fifth International Modelica Conference*, 2006.

Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet. A Conservative Extension of Synchronous Data-flow with State Machines. In *EMSOFT'05*, September 2005.

DO-178C. DO-178C Software Considerations in Airborne Systems and Equipment Certification, December 2011.

DO-330. DO-330 Software Tool Qualification Considerations, December 2011.

DO-332. DO-332 Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A, December 2011.

Julien Forget, Frédéric Boniol, David Lesens, and Claire Pagetti. A multi-periodic synchronous data-flow language. In *HASE 2008. 11th IEEE*. IEEE, 2008.

Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous dataflow programming language lustre. In *Proceedings of the IEEE*, 1991.

Atsushi Igarashi. *Formalizing Advanced Class Mechanisms*. PhD thesis, University of Tokyo, 1999.

Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight java: A minimal core calculus for java and gj. *ACM Trans. Program. Lang. Syst.*, May 2001.

Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7), 2009.

Modelica Association. Modelica – A Unified Object-Oriented Language for Systems Modeling, version 3.3. <http://modelica.org>, May 2012.

Marc Pouzet. *Lucid Synchrone, version 3. Tutorial and reference manual*. Université Paris-Sud, LRI, April 2006.

Lucas Satabin, Olivier Andrieu, Bruno Pagano, and Jean-Louis Colaço. Formalization of A Modelica Subset for Safety-Critical Software Development. Technical report, Esterel Technologies, 2015.

Bernhard Thiele, Stefan-Alexander Schneider, and Pierre R Mai. A Modelica Sub-and Superset for Safety-Relevant Control Applications. In *Proceedings of the Ninth International Modelica Conference*, 2012.

Fundamental EoS Implementation for {Water+Ammonia} in Modelica

Leonard Becker¹ José L. Corrales Ciganda¹

¹ Department of Energy Engineering, Technische Universität Berlin, Germany
jose.l.corralesciganda@tu-berlin.org, mail@leo-becker.de

Abstract

The implementation of a library for the calculation of thermodynamic properties for the mixture {water + ammonia} based on a fundamental equation of state (EoS) for the Helmholtz free energy is developed and presented. The model uses the formulation of Tillner-Roth and Friend (1998a) in order to provide the best available single state thermodynamic data. The calculation of the vapour-liquid equilibrium (VLE) using the fundamental equation of state is examined. However due to difficulties found under certain pressure and temperature conditions, another method for calculating the VLE had to be used. The problems found included unreliable results and difficulties setting the initial values. Saturation temperature polynomials by Johnson et al. (2001) have been found to be faster and more reliable and have been implemented instead. It's possible to calculate thermophysical properties in single and two-phase region at pressures from the melting point up to 40 MPa.

Keywords: Ammonia + Water, fundamental EOS, thermodynamic properties, Helmholtz energy

1 Introduction

The description of the thermodynamic properties of the involved substances is crucial in modelling an industrial process. Today, the most accurate EoS available are fundamental EoS in terms of Helmholtz energy. Using such an EoS it is possible to calculate all thermodynamic state properties. In addition the fugacity coefficients used to receive the VLE-states are available through the partial derivatives of the Helmholtz energy. Another popular approach is the use of polynomial fitted VLE-data. These explicit equations are quick to solve and also invertible.

The comprehensive CoolProp library by Bell et al. (2014) (included in ExternalMedia) has an implementation of the mixture {water + ammonia} as an incompressible fluid using polynomials. The ammonia content is also restricted to 30 % which limits use in sorption devices. In these devices the refrigerant has ammonia contents above 90 %. A fundamental EoS in terms of

Helmholtz energy is implemented in the HelmholtzMedia library by Thorade (2012) for single substance working fluids.

Carluccio et al. (2014) presented a simulation model for gas absorption heat pumps using {water + ammonia}. For this model the thermodynamic properties were calculated using correlations presented by Xu and Goswami (1999) based on the work of Ziegler and Trepp (1984). The formulation of the latter is based on a Gibbs Free Energy fundamental EoS and the measurement data used are older than those used by Tillner-Roth and Friend.

2 Formulation of the Helmholtz energy fundamental EoS

Tillner-Roth and Friend (1998b) performed a comprehensive assessment of available measurements on thermodynamic properties of the mixture {water + ammonia}. Using a formulation of the Helmholtz energy this resulted in the best available EoS regarding the accuracy and domain of definition (Tillner-Roth and Friend, 1998a).

The Helmholtz energy f is a function of temperature T , specific volume v and mole fraction x . In order to represent all influences on the Helmholtz energy it is divided into several parts. At first it is separated in ideal ϕ° and residual ϕ^r parts which both consist of a part for each substance and mixing terms. The Helmholtz energy is made dimensionless ϕ using the specific gas constant R and the temperature T and calculated in Equation (1).

$$\frac{f}{RT} = \phi = \phi^\circ(\tau, \delta, x) + \phi^r(\tau, \delta, x) \quad (1)$$

The arguments of the function are also made dimensionless using normalization temperature T_n and volume v_n calculated in Equations (2). In the ideal-gas part $T_n = 500 \text{ K}$ and $v_n = 15 \text{ kmol/m}^3$ are chosen arbitrarily. For the residual part reducing functions containing four parameters are introduced. These are needed for an accurate representation but are not introduced here.

$$\tau = \frac{T_n}{T}, \quad \delta = \frac{v_n}{v} \quad (2)$$

As explained above the ideal part ϕ° is split into pure substance and mixing sub-parts. In Equation (3) the logarithmic terms represent the ideal mixing terms. The mixing part $\Delta\phi^r$ of the residual part (Equation (4)) is fitted using up to 56 parameters.

$$\phi^\circ = (1-x)\phi_{01}^\circ + x\phi_{02}^\circ + (1-x)\ln(1-x) + x\ln x. \quad (3)$$

$$\phi^r = (1-x)\phi_1^r + x\phi_2^r + \Delta\phi^r \quad (4)$$

The modular structure allows the use of proven formulations for the residual parts of the pure substances. For water IAPWS from Wagner and Pruß (2002), which is already implemented in the Modelica Standard Library, is used. Ammonia properties are approximated with the formulation from Tillner-Roth et al. (1993).

All the parameters and equations, together with an assessment of the database may be found in the original article from Tillner-Roth and Friend (1998a).

2.1 Calculating Auxiliary Properties

In order to calculate any auxiliary property at first the Helmholtz energy with its arguments has to be known (see Section 4.3). To get the thermodynamic properties from the Helmholtz energy partial derivatives are used. The algorithms shown in Table 1 are derived from the total differential of the Helmholtz energy and the property in question. The process is described in detail in Thorade and Saadat (2013). Here one may also find a derivation of several partial derivatives of the Helmholtz energy in the one- and two-phase region.

As shown in Table 1, partial derivatives with respect to the molar fraction are only needed for the algorithms calculating the fugacity for both substances.

Table 1. Equations for auxiliary properties

Property	Equation
pressure	$p = RT\rho \left(1 + \delta \frac{\partial \phi^r}{\partial \delta} \right)$
inner energy	$u = RT \left(\tau^\circ \frac{\partial \phi^\circ}{\partial \tau^\circ} + \tau \frac{\partial \phi^r}{\partial \tau} \right)$
enthalpy	$h = RT \left(1 + \tau^\circ \frac{\partial \phi^\circ}{\partial \tau^\circ} + \tau \frac{\partial \phi^r}{\partial \tau} + \delta \frac{\partial \phi^r}{\partial \delta} \right)$
entropy	$s = R \left(\tau^\circ \frac{\partial \phi^\circ}{\partial \tau^\circ} + \tau \frac{\partial \phi^r}{\partial \tau} - \phi^\circ - \phi^r \right)$
specific heat	$c_v = -R \left(\tau^{\circ 2} \frac{\partial^2 \phi^\circ}{\partial \tau^{\circ 2}} + \tau \frac{\partial^2 \phi^r}{\partial \tau^2} \right)$
fugacity coef.	$\varphi_{ammonia} = \frac{\exp(\phi^r + \delta \frac{\partial \phi^r}{\partial \delta} - xF_\varphi)}{1 + \delta \frac{\partial \phi^r}{\partial \delta}}$
	$\varphi_{water} = \frac{\exp(\phi^r + \delta \frac{\partial \phi^r}{\partial \delta} - (1-x)F_\varphi)}{1 + \delta \frac{\partial \phi^r}{\partial \delta}}$
fugacity var.	$F_\varphi = \frac{\partial \phi^r}{\partial x} + \frac{\partial \delta}{\partial x} \frac{\partial \phi^r}{\partial \delta} + \frac{\partial \tau}{\partial x} \frac{\partial \phi^r}{\partial \tau}$

3 Vapour-Liquid Equilibrium

3.1 Solving the PDE in Modelica

The vapour-liquid equilibrium for the mixture {water + ammonia} is defined through the three equilibria:

$$\text{mechanical equilibrium: } p_{vapour} = p_{liquid} \quad (5)$$

$$\text{thermal equilibrium: } T_{vapour} = T_{liquid} \quad (6)$$

$$\text{chemical equilibrium: } \tilde{G}_{NH_3,vapour} = \tilde{G}_{NH_3,liquid} \quad (7)$$

$$\tilde{G}_{H_2O,vapour} = \tilde{G}_{H_2O,liquid} \quad (8)$$

For VLE-calculations equations (5) to (8) have to be used in combination with the EoS formulation of Helmholtz energy. To get thermodynamic information in the VLE-region temperature, specific volume and molar fraction for both phases are needed. With these values in hand the other state properties may be calculated using the algorithms from Section 2.1. Six variables are to be found and four equations given resulting in a algebraic equation system with a degree of freedom of two. In practice two variables out of temperature, pressure and molar fraction of the phases are used. Several approaches to solve the VLE in an efficient and robust way can be found in the literature, like in Privat et al. (2013). In this first approach however the set of equations are written with Modelica and the system solved providing two inputs variables and using standard compiler and solvers.

The chemical equilibrium in Equations (7),(8) may be converted to the chemical potential μ for the species k using the definition of the partial molar Gibbs energy \tilde{G} . Equation (9) is converted into Equation (10) to enable the use of fugacity coefficients φ provided by the EoS

$$\mu_{k,vapour} = \mu_{k,liquid} \quad (9)$$

$$x_{k,vapour}\varphi_{k,vapour} = x_{k,liquid}\varphi_{k,liquid} \quad (10)$$

For a given couple of temperature and pressure the VLE has been computed using the PDE resulting from the EoS with a Modelica model solved with the DASSL solver. As shown In Figure 2 the solver is not able to find a solution under certain combinations of T and p.

3.2 Polynomial Saturation Curves

A widely used approach to approximate VLE-data is fitting with polynomial saturation curves. Patek and Klomfar (1995) introduced a methodology which is refined by Johnson et al. (2001). The measurement datasets used for the formulation of the Helmholtz energy are also used here which results in similar results. The temperature, pressure and molar fraction data is forming a three dimensional surface, which is fitted using a weighted least square algorithm. The result are saturation temperature curves $T_{phase}(p, x)$ with the form of Equation (11) using

56 parameters.

$$T_{phase}(p, x) = t_0 \sum_i a_i x_{phase}^{q_i} (1 - x_{phase})^{r_i} \left[\ln \left(\frac{p_0}{p} \right) \right]^{s_i} \quad (11)$$

The polynomials are build as Lagrange polynomials to ensure pure substance saturation lines equal to Equations (12) and (13).

$$T_{H_2O}(p) = 269.8p^{0.08839} + 52.79p^{0.3663} + 130.4 \quad (12)$$

$$T_{NH_3}(p) = 177.9p^{0.09397} + 40.28p^{0.3898} + 79.83 \quad (13)$$

The equations are taken from Reynolds (1979). Oscillation can be problematic in numeric algorithms. This behavior of the fitted polynomials was reduced near the pure substance data. Multiple new datasets are generated to punish oscillation in the least square method. The methodology in detail and the parameters of Equation (11) may be found in Johnson et al. (2001).

4 Modelica Implementation

The general implementation of the library `H2O_NH3_TillnerRoth` is shown in Figure 1 and is mostly based on the Modelica Standard Library. To calculate fluid properties the `model BaseProperties` may be used. Here all basic properties are included for easy use. All the functions can be accessed independently, too. The folder `VLE` includes the VLE computation using the polynomial saturation curves discussed in Section 3.2. Some tests may be carried out using the functions contained in the `package Test` (see Sec. 4.4).

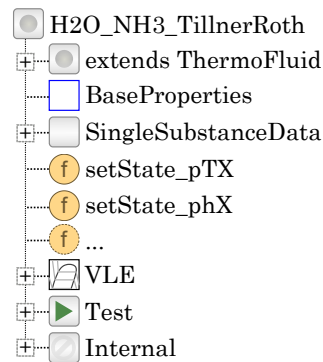


Figure 1. Basic structure

The implemented library `H2O_NH3_TillnerRoth` provides functions and models for the calculation of thermodynamic properties and vapour-liquid equilibrium of ammonia-water mixtures. For the VLE calculation the library makes use of the polynomial saturation curves previously presented. It is also possible to solve the VLE using the Helmholtz energy fundamental EoS.

A new interface `Thermofluid` is introduced, that extends the standard `PartialMedium` interface. To ensure compatibility all of the defined functions and records are used. In `Thermofluid` all the newly introduced registers are implemented. For example the `record FluidConstants` is extended with the data, that are needed for the dimensionless expression of the variables in the fundamental equation of state, such as density and molar mass at critical points.

To calculate fluid properties in one phase several steps are taken successively:

1. At first the thermodynamic state vector (d, T, x) has to be determined. This is described in Section 4.3. The state vector is assigned to the `record ThermodynamicState`, which contains the arguments of the Helmholtz energy, in their mass based form.
2. As input for the Helmholtz energy EoS the state vector has to be made dimensionless in the `function tauDeltax_`.
3. Actual partial derivatives (Section 4.1) can now be calculated. The algorithms are found in the `package Internal.EOS` and assigned to the `record TauDeltaX`.
4. Lastly the high-level functions defined in the Modelica Standard Library access the derivatives to calculate the thermodynamic property as described in Section 2.1.

The implementation for the EoS is based on the ‘HelmholtzMedia’ library presented by Thorade (2012). Some modifications are needed however due to the fact that HelmholtzMedia was developed for single substances and not mixtures. This also adds an extra independent variable for the EoS, changing the definition of the thermodynamic state vector and the system of equations for the VLE.

4.1 Partial derivative calculation

The partial derivatives of the formulation needed for the algorithms have been implemented using two different approaches. Firstly, the Modelica feature of automated symbolical derivatives is used. This is a very convenient solution, producing little program code and effort to implement. The implementation is described in Olsson et al. (2005) for single substance use. The partial derivative of the residual part of the Helmholtz energy $\frac{\partial \phi^r}{\partial \delta}$ is calculated using

```
function dphir_ddelta = der(phir_, delta);
```

with the `function phir_` from Equation 4 and the argument `delta`.

However the use of this feature has not been shown as completely satisfactory. The calculation of the derivatives is not reliable since errors often appear caused by division by zero in logarithmic expressions. It was not possible to find a way to avoid these errors, because the algorithm used to automatically calculated the derivatives was not known, not manipulable and beyond the scope of this work.

Finally a second approach was used, implementing analytically determined derivatives. The formulation of

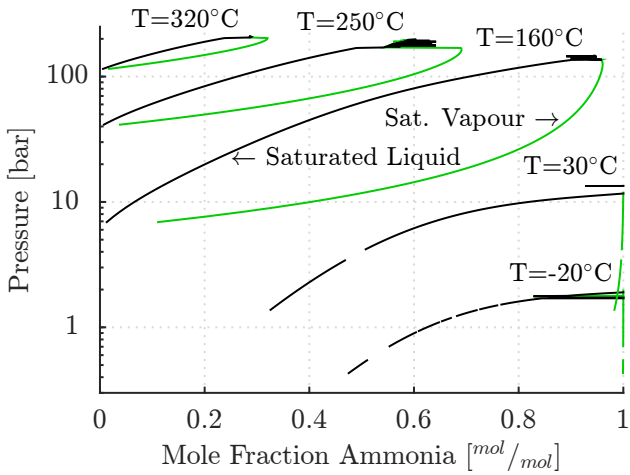


Figure 2. Solution of the VLE in Modelica for different temperatures

the Helmholtz energy is derivable analytically due to its structure as mostly a sum of small terms. With this second method the errors can be avoided through substituting the unfeasible terms with zero. The price is more complex Modelica code and the work load for the model maker associated with analytically solving the derivatives.

4.2 Vapour-Liquid-Equilibrium Calculation

As shown in Figure 2, the use of the DASSL solver to solve the equation system of the VLE resulting from the EoS formulation of Helmholtz Energy is not successful in certain regions. Especially for low pressures (below 1 bar) and for mole fractions close to the single substances is not possible for the solver to find a solution.

In addition the solver is only able to solve the PDE if the given start values are very close to the solution. This can become a problem for dynamic simulations with big changes in pressure, temperature or mole fractions, since the start values have to be supplied to the model as parameters. Computed data from other components or former time steps is stored as variables. In the current version of Modelica parameters can't be calculated using variables. Not being able to use simulation data makes it very difficult to supply good start values to the VLE equation system.

For these reasons, the alternative implementation using the polynomial saturation curves is preferred, since it shows no convergence problems for the whole working region and the deviation of its results from those of the EoS implementation is in the range of measurement error reported by Tillner-Roth and Friend (1998a).

4.3 Finding the Thermodynamic State

As outlined by Thorade (2012) in engineering applications often the known variable combinations are (p,T,x)

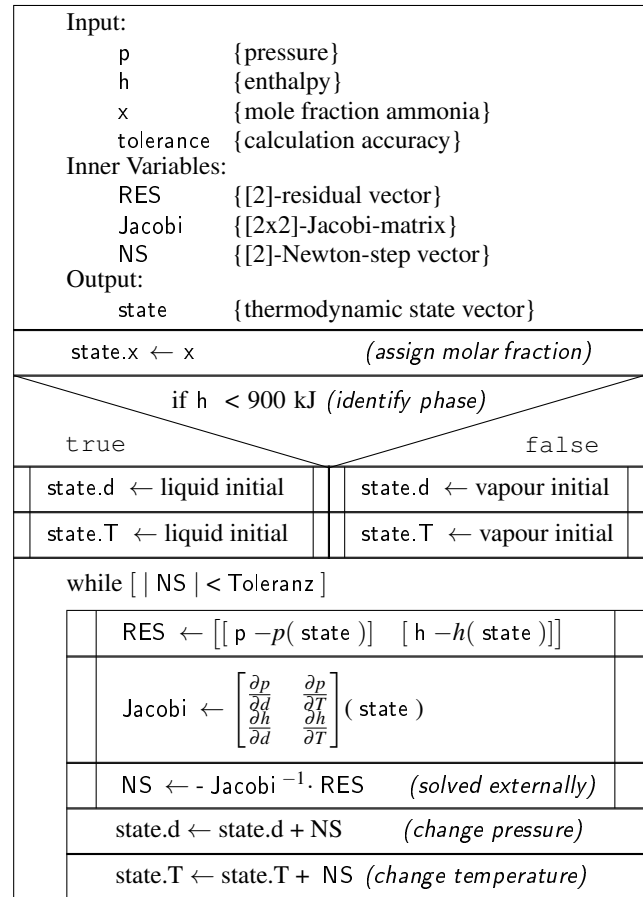


Figure 3. Nassi-Shneiderman-Diagram of setState_phx

or (p,h,x) instead of (d,T,x), the arguments of the Helmholtz energy. This is particularly important for users of the Modelica Fluid library, since its connectors share the combination (p,h,x) between components.

In the Library H2O_NH3_TillnerRoth the functions setState_pTx and setState_phx are implemented. These functions iteratively determine a thermodynamic state (d,T,x) starting from (p,T,x) or (p,h,x).

The Nassi-Shneiderman-Diagram for the function setState_phx is shown in Figure 3 as an example. After the unchanged mass fraction is assigned the phase needs to be determined. In the case of given (p,h,x) this can easily be achieved, because in the area of definition the enthalpy identifies the phase unambiguously. Afterwards a two-dimensional Newton-Raphson method is executed.

Firstly the input pressure is compared to the pressure computed by the current iteration of the state vector. Together with the analogue difference for the enthalpy the residual vector is assigned. In the Jacobi matrix the partial derivatives of pressure and enthalpy with respect to temperature and density are calculated using the current state vector. The equation $NS = -Jacobi^{-1} \cdot RES$ is rearranged to $Jacobi \cdot NS = RES$ and solved with the Gaussian elimination implemented in the Modelica Standard Library to receive the Newton step vector NS. Finally the

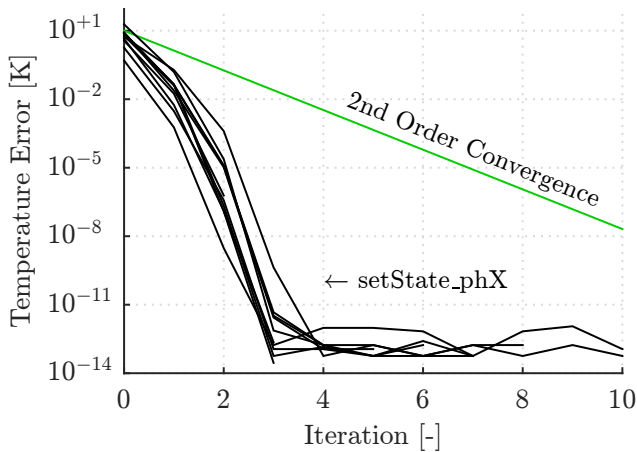


Figure 4. Temperature convergence of `getState_phx`

state variables temperature and density are changed.

Convergence behaviour

To test the reliability and speed of the `getState` functions convergence examinations are carried out. A list of 27 sample points in both phases and different temperatures and pressures was used. In Figure 4 the difference of the calculated temperature and the analytic solution is plotted on the iteration. It can be seen that the convergence is higher than the upper bound of the Newton-Raphson method, convergence of second order.

Having said that, for temperature errors below 10^{-12} the method is not able to converge any more in some cases. These artefacts have been removed through a capping of the tolerance. In result the `getState` functions are able to find a solution in mostly three iterations with an error lower to the inaccuracies of the formulation in question.

4.4 First Evaluation using PartialTestModel

A first test of the usability and performance of the Library was undertaken using the `PartialTestModel` from the Modelica Standard Library. The model shown in Figure 5 includes models for a source, volume, pipe and sink and follows a principle recommended by Tummescheit (2002). In Equation (14) the start conditions of the system is defined. For the source the values are slightly different (Eq. (15),(16)) which are then ad-

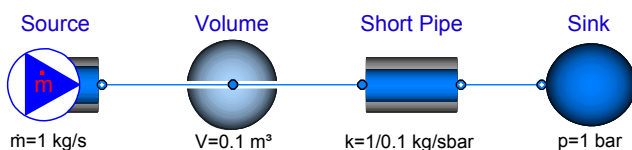


Figure 5. Diagram view of the `PartialTestModel`

ditions of the system is defined. For the source the values are slightly different (Eq. (15),(16)) which are then ad-

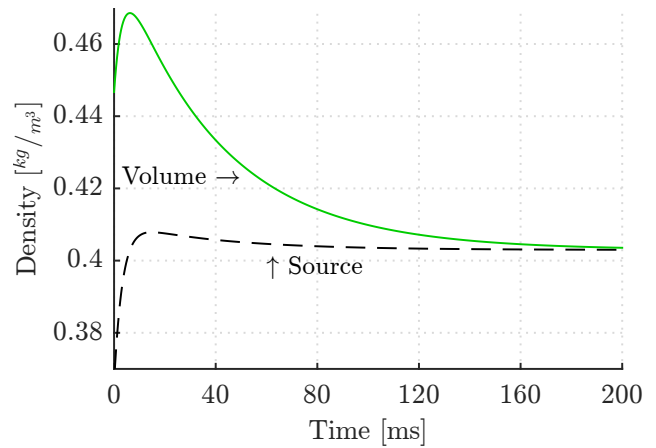


Figure 6. Dynamic simulation of `PartialTestModel`: density

vanced in the system through the mass flow.

$$p_{start} = 1 \text{ bar}, \quad T_{start} = 200^\circ\text{C}, \quad x_{start} = 0.5 \quad (14)$$

$$T_{Source} = 1, 2T_{start} = 294, 8^\circ\text{C} \quad (15)$$

$$x_{Source} = 0, 5x_{start} = 0, 25 \text{ kg/kg} \quad (16)$$

The results from Figure 6 and 7 show that the model is able to calculate thermodynamic states for dynamic changing conditions.

In Figure 6 the density in the simulated volume is plotted. The density of the fluid in the volume changes with changing pressure and temperature. The changes of pressure and temperature with time are shown in Figure 7.

The temperature of the volume changes from the starting 200°C since more and more fluid is flowing into the volume from the source with a constant temperature of $\sim 300^\circ\text{C}$. The pressure in the volume changes dynamically when fluid starts flowing until a stationary state is found (the source provides a constant mass flow rate). The density rises at the beginning as a consequence of the increasing pressure, but afterwards for constant pressure decreases with increasing temperature. The obtained results follow the expected behaviour for such a system.

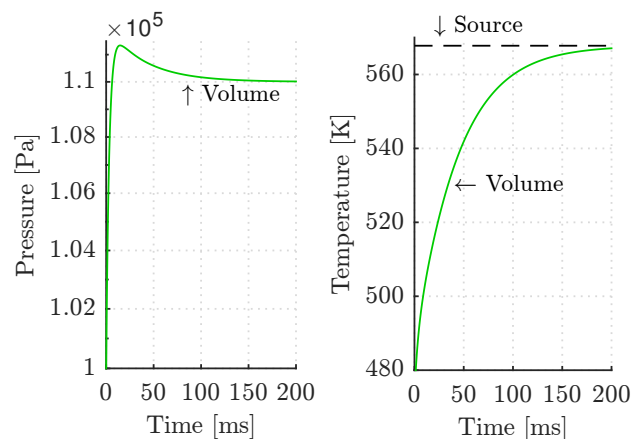


Figure 7. Dynamic simulation of `PartialTestModel`: pressure and temperature

5 Summary and Outlook

A Modelica library for the calculation of thermodynamic properties of the mixture ammonia-water has been presented and discussed. The formulation of the Helmholtz energy fundamental equation of state using Modelica has been demonstrated. For the partial derivatives needed in the EoS, the automated symbolical derivation implemented in Modelica has been tested with not reliable results, and previously analytically solved differentials has been used instead.

For the solution of the vapour-equilibrium equations two alternative approaches have been tested. A first approach, solving the resulting PDE with Modelica has been found possible but inconvenient, since it requires quite accurate start values, takes long CPU-calculation times and is not able to solve the equation system under certain conditions. Instead, a second approach using polynomial saturation curves is preferred, and has been tested with regards of convergence using different methods.

In future works, the library will be extended to include calculation of non-state properties like surface tension, viscosity or thermal conductivity. that are relevant for the simulation of absorption heat pumps and chillers. Possible improvements can be made regarding the automatic recognition of the fluid phase to improve initialisation and simulation speed. An algorithm for the calculation of the VLE PDE could also be implemented.

A new Modelica Media connector for multiple phase mixtures (similar to FluidPort) would be very useful for heat transfer simulations, and would also require the adaptation of the presented library. The results obtained with this library are still to be compared with those obtained by the correlation equations implemented by Carluccio et al. (2014) or those obtained accessing external data base properties like REFPROP from Lemmon et al. (2013) or CoolProp from Bell et al. (2014).

References

- Modelica Association et al. The modelica standard library - 3.3 revision 1. Online, URL: <http://www.modelica.org/>, 2014.
- I.H. Bell, J. Wronski, S. Quoilin, and V. Lemort. Pure and pseudo-pure fluid thermophysical property evaluation and the open-source thermophysical property library coolprop. *Industrial & Engineering Chemistry Research*, 53(6):2498–2508, 2014. doi:10.1021/ie4033999.
- F. Carluccio, G. Starace, and C. Bongs. Modeling and simulation of a gas absorption heat pump: Proceedings ishpc 2014, 106. 2014.
- D. U. Johnson, W.E. Lear, and S. A. Sherif. *Curve Fitting of Ammonia-Water Mixture Properties: An Improvement of Patek and Klomfar's Ammonia-Water Correlations*. PhD thesis, University of Florida, 2001.
- EW Lemmon, ML Huber, and MO McLinden. Reference fluid thermodynamic and transport properties-refprop, version 9.1. *National Institute of Standards and Technology Standard Reference Database 23*, 23, 2013.
- H. Olsson, H. H Tummescheit, and H. Elmqvist, editors. *Using automatic differentiation for partial derivatives of functions in Modelica*, 2005. Citeseer.
- J. Patek and J. Klomfar. Simple functions for fast calculations of selected thermodynamic properties of the ammonia-water system. *International Journal of Refrigeration*, 18(4):228–234, 1995. ISSN 01407007. doi:10.1016/0140-7007(95)00006-W.
- R. Privat, J.N. Jaubert, and Y. Privat. A simple and unified algorithm to solve fluid phase equilibria using either the gamma-phi or the phi-phi approach for binary and ternary mixtures. *Computers and Chemical Engineering*, 50:139 – 151, 2013. ISSN 0098-1354.
- W. C. Reynolds. Thermodynamic properties in si. graphs, tables and computational equations for forty substances, department of mechanical engineering, 1979.
- A. Thorade, M. and Saadat. Helmholtzmedia - a fluid properties library. In *Proceedings of the 9th International Modelica Conference*. doi, volume 10, page 3384, 2012.
- M. Thorade and A. Saadat. Partial derivatives of thermodynamic state properties for dynamic simulation. *Environmental Earth Sciences*, 70(8):3497–3503, 2013. ISSN 1866-6280.
- R. Tillner-Roth and D. G. Friend. A helmholtz free energy formulation of the thermodynamic properties of the mixture {water + ammonia}. *Journal of Physical and Chemical Reference Data*, 27(1):63–96, 1998a. ISSN 0047-2689.
- R. Tillner-Roth and D. G. Friend. Survey and assessment of available measurements on thermodynamic properties of the mixture {water+ ammonia}. *Journal of Physical and Chemical Reference Data*, 27(1):45–61, 1998b. ISSN 0047-2689.
- R. Tillner-Roth, F. Harms-Watzenberg, and H. D. Baehr. Eine neue fundamentalgleichung für ammoniak. *DKV TAGUNGSBERICHT*, 20:67, 1993. ISSN 0172-8849.
- H. Tummescheit. *Design and implementation of object-oriented model libraries using modelica*. PhD thesis, Lund University, 2002.
- W. Wagner and A. Pruß. The iapws formulation 1995 for the thermodynamic properties of ordinary water substance for general and scientific use. *Journal of Physical and Chemical Reference Data*, 31(2):387–535, 2002. ISSN 0047-2689.
- Feng Xu and D.Yogi Goswami. Thermodynamic properties of ammonia-water mixtures for power-cycle applications. *Energy*, 24(6):525 – 536, 1999. ISSN 0360-5442.
- B Ziegler and Ch Trepp. Equation of state for ammonia-water mixtures. *International Journal of Refrigeration*, 7(2):101 – 106, 1984. ISSN 0140-7007.

MultiComponentMultiPhase – a framework for thermodynamic properties in Modelica

Johan Windahl¹ Katrin Prölss¹ Maarten Bosmans² Hubertus Tummescheit¹ Eli van Es² Awin Sewgobind²

¹Modelon AB, Lund, Sweden,

{johan.windahl, katrin.prolss, hubertus.tummescheit}@modelon.com

²VORtech, Delft, Netherlands, {maarten.bosmans, eli.vanes, awin.sewgobind}@vortech.nl

Abstract

This paper describes the development and requirement specification of an open-source framework for multi-phase multi-component thermo properties in Modelica. The goal is to have a standardized interface to multi-component multi-phase fluids with access to external property packages in Modelica. This will make it easier to develop models for e.g. the process industry. The library uses a model based interface and implications of such a design are analyzed and compared with the traditional function based interface.

The work has been carried out in collaboration with Modelon AB and VORtech within the umbrella of *Methods and tools* as part of the CleanSky SGO project.

Keywords: CAPE-OPEN, FluidProp, RefProp, fluid properties, flash calculations

1 Introduction

Properties of working fluids define the achievable baseline accuracy for fluid system simulations. The availability of properties for steam and flue gases initiated the use of Modelica in the power industry, where it today is a well-established technology with several commercial and open source libraries available (Modelica Libraries, 2015). High quality fluid properties are laborious to produce and their non-availability is therefore a typical blocking argument for the use of a certain tool or technology.

Published work of modeling chemical process systems in Modelica exists, see (Tummescheit *et al*, 2002; Dietl *et al* 2011; Baharev *et al* 2012). But until today, the use of Modelica has not been widely spread to this area even though it would be well-suited to describe these processes. Modelica is equation based, similar to gPROMS, which is well established in the process industry. However, it is lacking a standardized interface for multi-component multi-phase fluid properties. For an overview of equation oriented methods for chemical and related process flowsheets, see (Morton, 2002).

In this project a Modelica library for multi-phase multi-component fluids has been developed together with an external C/C++ Modelica property interface with back ends to CAPE-OPEN, RefProp (Lemmon *et al*, 2013) and FluidProp (Fluid property library, 2015). The framework also contains a Modelica library for distillation processes for verification and testing of the media interface design.

2 Background

Modelica.Media is a freely available Modelica package contained in the Modelica standard library. It consists of property models from ideal gases up to high accuracy models of WaterIF97 and R134a. The current version 3.2.1 is restricted to pure two-phase or single phase mixtures. The goal is to extend the capabilities of Modelica.Media with support of multi-component multi-phase mixtures and to find an interface structure that is user-friendly both from an implementer and end-user perspective.

In order to collect input for the interface design, a meeting in Delft (Oct 2013) was held that gathered 17 people from academia, industry and members of the Modelica design group. During the meeting it became clear that technical challenges to implement such a property interface efficiently using the current structure of Modelica.Media are high. A large part of the project has therefore been focused on finding an interface structure both within and outside the limits of the Modelica specification 3.3.

2.1 Application overview

The framework developed in this project must cover a wide range of processes. The following types of processes have been identified where multi-phase multi-component fluids are used:

2.1.1 Thermodynamic cycle processes

Typical applications are refrigeration, heat pumps (vapor compression cycles) and Organic Rankine cycles using a blend of different working fluids, which may be available in RefProp. Models are usually characterized by a homogenous treatment of properties and flows, usually modelled with mass based units as is common in the energy and power industry. System may operate in overcritical conditions.

These applications fit well into the Modelica.Media structure as it was created with these processes in mind. Flexible models are required due to the number of present phases in a model can change during simulation.

2.1.2 Thermal separation processes (with and without chemical reactions)

Typical applications are rectification and absorption processes where the numbers of phases usually are limited to vapor and liquid and models use a mole basis. It is common to include chemical reactions and fluids are often taken from databases via CAPE-OPEN.

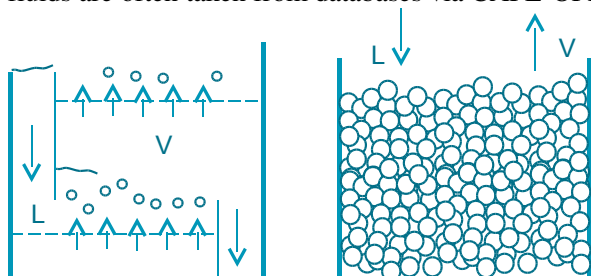


Figure 1. Example of thermal separation processes. Column tray (left) and column packing (right).

2.1.3 Transport of multiphase flows

Typical applications are compositional pipe network simulations, which are computationally expensive. In this case multiple phases may coexist and a homogenous assumption is not valid.

2.2 Context

Following types of usage are possible:

- Dynamic simulation
- Steady-state simulation
- Optimization

From an interface perspective a difference is in the requirement of differentiation of properties. Solving an optimal control problem also involves the Hessian (Boyd *et al*) Even if these can be calculated numerically the performance and robustness increase if analytical derivatives can be provided (Åkesson *et al*, 2012) An interface should therefore support the usage of analytical derivatives if these can be provided.

2.3 Phase equilibrium calculation

A phase equilibrium calculation determines subject to specified constraint, e.g. fixed pressure and temperature, present phases and the composition and fraction of each present phase. It is an iterative calculation which often uses specialized algorithms; see (Parekh *et al*, 1998; Gernert *et al*, 2014).

Phase equilibrium calculations are time consuming and will dominate the total CPU usage, up to 95% according to (Trapp, 2014). Similar numbers have also been observed in this work.

To achieve competitive performance:

1. The number of phase equilibrium calculations should be minimized. This can be achieved by designing fluid and application library interfaces so that calculation result can be shared between components. This may require expanding the connector class with additional variables to avoid redundant calculations.
2. For each phase equilibrium calculation, the number of iterations inside these algorithms needs to be minimized. This can be achieved by providing good iteration guess values.

3 Modelica media interface

There are several possibilities to define an interface due to Modelica support both models and functions.

The first step in the design process was to analyze and list requirements.

3.1.1 General media requirements

1. User-friendliness and structure
 - The structure should be easy to understand and use. Implementation details such as external objects should be hidden from the user.
 - Interface that can be used by both a native Modelica and external C-code media implementation.
 - Calculation of parameters, preferable also structural parameters, from property functions that may have a dependency on external code/external object.
2. Possibility to create a media structure using inheritance. To easily create new media from existing templates and interfaces.
3. Performance
 - The interface should be designed with efficiency in mind. It should support differentiation of properties and caching through external objects.
 - Possibility to provide additional information about present phases that can be used to simplify or skip computational expensive phase equilibria calculations.

- Support to add initial guesses of start values.
- Derivative functions
 - It should be possible to specify derivative functions needed for: state variable transformations, index reduction and generation of analytic Jacobians.
 - Additional
 - It should be possible to extend the interface with new functionality such as reaction properties.

3.1.2 Multi-component multi-phase requirements

There is a wide range of different properties that may be needed but here we consider the most basic usage.

- Calculation of phase equilibrium.
- Calculation of properties for a specified phase at phase equilibrium.
- Support for common properties such as fugacity and activity coefficients.
- Support of both molar and mass based properties. The chemical process industry usually works in mole while the energy industry works in mass based units.
- No restriction of the number of supported phases.

3.1.3 Additional requirement

- Take advantage of unit declaration. The possibility to declare units is a powerful feature in Modelica that should be used.
- Uniquely identify phases and compounds.

3.2 Design restrictions

When designing a property interface in Modelica following restrictions (Modelica Association, Specification, 2015) needs to be considered.

- Functions need to be pure, i.e. they are not allowed to have any side effect. This is a fundamental assumption in Modelica that makes it possible for a tool to apply symbolic transformation and rearrange calculations.
- Records are not suited to be used as function inputs. This is due to the fact that it is not allowed to specify a derivative function if the record contains a non-real variable. It is also not efficient to use a record with additional variables due to all variables needs to be considered for differentiation. A record is also not allowed to contain an external object.
- It is not possible to access a previous value of a continuous variable. There is no such operator in Modelica.

3.2.1 Iterative algorithms

A consequence of the restrictions is that it is inefficient to implement explicit iterative algorithms in native Modelica. This is due to that functions are not allowed to have internal memory between function calls and there is no operator to access a previous value of a continuous variable. The start value of a variable in an algorithm is therefore equal to its start attribute during simulation. This is a drawback for function based media property calculations that need to be solved by an iteration process. If instead a model based interface is used and the algorithm is replaced with an implicit equation formulation, the tools non-linear solver can use the last solution point as a start for the next iteration (Olsson *et al*, 2005).

3.3 Model based interface structure

Based on the requirements and restrictions it was decided that the interface structure should be model based (this does not hinder the implementation to be function based).

Main advantages with a model based structure are:

- Possible to implement a medium using a declarative approach as demonstrated in (Olsson *et al*, 2005). It makes it possible to quickly create a medium with good performance, see section 5.1
- Possible to share interface between an external code based media and a native Modelica based media.
- User friendly as implementation details can be hidden from the user and the possibility to work graphically and by that taking advantage of a tool support of e.g. unit conversion.
- Possible with a minimalistic interface as it is not necessary to create new models for new input combinations. The model based interface may not specify the causality of the variables.
- Avoid the need for a tool to support common-sub expression elimination as the result of an expensive calculation can be stored in a model.
- Possible to use block and models in an implementation, e.g. the Modelica Standard Library tables.

There are also disadvantages that should be considered:

- A model can't be instantiated inside a function which limits the scope where a media calculation can be executed.
- It is user unfriendly to calculate parameters from a media. It requires setting a parameters fixed attribute to false and an additional equation in the initial equation section.

- Not possible to calculate a property on demand as a model needs to be instantiated.

3.3.1 MultiPhaseMixture interface

The current interface structure is shown in **Figure 2**, it consists of a few models and helper functions.

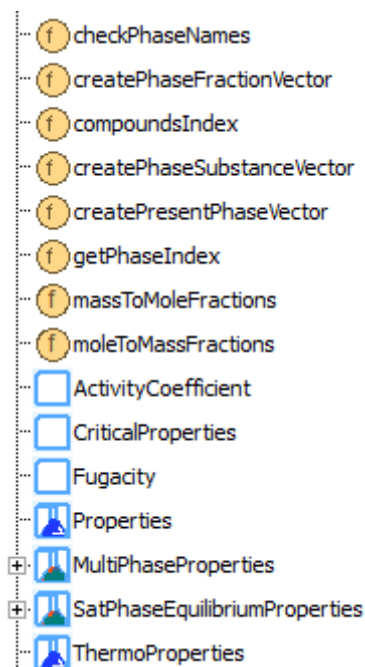


Figure 2. Screenshot of the current interface structure of MultiPhaseMixture. *Note that it is currently under development and is therefore subject to change.*

3.3.2 Example of usage

An example of how to use the ThermoProperties model is shown in Listing 1.

```
model ExampleOfUsage
package Medium=MultiPhaseMixture.Air_PureModelica;
Medium.ThermoProperties thermoProperties(
  inputs=MultiPhaseMixture.Interfaces.Inputs.pTY,
  p=100000,
  T=298.15,
  Zm=Medium.reference_Y);
Density d;
equation
d= thermoProperties.d;
end ExampleOfUsage;
```

Listing 1. Modelica code showing the usage of a property calculation using the model based interface.

4 External interface

Developing thermodynamic property models for multi-phase multi-component fluids is fairly complex and requires specialist knowledge. There already exist tools like MultiFlash and FluidProp that have been developed in the process industry and in academia over

a long time. It is therefore useful for the new Media library to be able to interface with external fluid property tools and databases.

The overall structure of the external framework is illustrated in **Figure 3**.

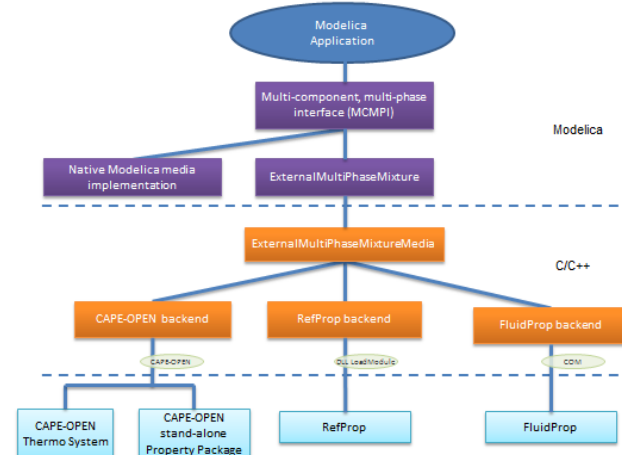


Figure 3. Overview of external framework structure.

4.1 Previous work in Modelica

There exists previous published work with interfacing external properties in Modelica, see (Tummescheit, 2002; Trapp 2014; Wellner 2014). There is also an open-source framework available, ExternalMedia (Casella, 2008), but it is limited to pure two-phase media.

4.2 Using external code in Modelica

Modelica has an external function interface to C which makes it possible to use external routines within a Modelica function. Following issues have been considered when building the new Modelica libraries around external code:

- Differentiation
- Error messages
- Use of external objects

An advantage of using native Modelica code over external code is that the Modelica compiler has access to structural information on the dependency between inputs and outputs. This makes it possible to automatically differentiate, create analytical Jacobians and explore sparsity patterns that will increase robustness and performance of a simulation.

For external functions, derivative information needs to be specified by the user. In the general case it would require full knowledge about the implementation. For thermodynamic properties the effort on providing this information can be decreased by taking advantage of thermodynamic properties definitions. But it is important that there are test cases as wrongly

calculated derivatives will lead to convergence failure which may be very hard to debug.

Another important aspect is implementation of appropriate error messages, as otherwise the simulation may crash without leaving any information to the user.

4.2.1 External object

With the use of external objects there is a defined way to allocate and de-allocate resources. It is also possible to store internal information between function calls which may be used to cache iteration start values. The C-interface is therefore based on external objects.

Disadvantages are restrictions on how they can be used in Modelica (Modelica Spec, 2015) and it is not clear how they work in combination with symbolic transformation such as inverse functions and sub-expression elimination. We have also encountered bugs related to the use of them, however it seems more stable with more recent versions of Modelica tools.

4.3 Modelica external media interface

MultiPhaseMixture.ExternalMixture is a template that implements the Modelica MultiPhaseMixture interface.

It consists mainly of:

- Functions that call the external C- code interface.
- Wrapper functions for various property and input combinations.
- Calculations of multi-phase properties and derivatives

With the wrapper functions it is possible to specify derivative annotations and support differentiation of external properties.

4.3.1 Derivatives

Neither CAPE-OPEN nor RefProp supports total overall properties derivatives, which may be needed for dynamic simulation, especially for state variable transformation. It was therefore decided that these types of calculations should be implemented on the Modelica side and not in the C-interface. For a pure fluid, the calculations are straight-forward and there are publications available (Thorade *et al*, 2013). For mixtures they are more complicated (Li, 1955). The difficult part is when several phases coexist. In that case they are currently calculated numerically.

```
function density_derh_p
  input Properties state;
  input MExternalMediaObject eo "External object";
  output Real ddhp "Density derivative wrt h at constant pressure";
  protected
  ...
  algorithm
    if (state.nbrOfPresentPhases == 1) then
      pd:=state.dpd_TN_1ph[integer(state.presentPhaseIndex[1])];
      ...
      ddhp:= -
state.d_overall*state.d_overall*pt/(state.d_overall*state.d_overall*pd*
cv + state.T_overall*pt*pt);
      elseif (nC == 1 and nP== 2) then
        ...
        dpT := (vap_s - liq_s)*liq_d*vap_d/(max(liq_d - vap_d,eps));
        ddhp:=state.d_overall*state.d_overall/(dpT*state.T_overall);
      else
        // multiple phases - calculate ddhp numerically
        d_deltah:= Wrapper_phX.density_phX(p=state.p_overall, h=state.h_o
verall+deltah,X=X,
state=calcProperties_phX(p=state.p_overall,h=state.h_overall+deltah,
X=X,eo=eo),eo=eo);
        ddhp:=(d_deltah-state.d_overall)/deltah;
      end if;
```

Listing 2. Code snippet of a Modelica implementation of density derivative wrt to specific enthalpy at constant pressure.

```
protected
  Auxiliary.Properties properties;
  final parameter ExternalMediaObject eo=
ExternalMediaObject(setupInfo);
  equation
    if (inputs == Inputs.pT) then
      properties =Auxiliary.calcProperties_pTX(p=p,T=T,X=Z,eo=eo);
      d =Auxiliary.Wrapper_pTX.density_pTX(p=p,T=T,X=Z,
state=properties);
```

Listing 3 Code snippet of the MultiPhaseProperties model for the external media template.

4.3.2 Mole vs mole fractions

A recommendation was given to use mole numbers instead of mole fractions (Szczepanski, 2013). The sum of all mole fractions is equal to 1:

$$\sum_{i=1}^N x_i = 1 \quad (1)$$

Mole fractions are not independent and are therefore more difficult to differentiate. This is further explained by (Molerup *et al*, 2002) where they state “*Derivatives with respect to mole fractions are best avoided, as they require a definition of the ‘dependent’ mole fraction and in addition lead to more complex expressions missing many important symmetry properties.*”

What is not described is how this can be applied to a thermodynamic framework for dynamic simulation which may contain intensive continuous state variables or connector variables.

Implications of changing the input mole fraction vector to a molar substance vector in a function interface are:

- A media model that is written in intensive form needs to add an extra conversion inside all functions. There may be cases where there are transformations back and forth. This affects the performance as it makes automatic differentiated code more complicated. This was seen in a simulation of the DistillationColumn model using the native Modelica media, where the total CPU time was increased with 7% when the input to the fugacity was changed from mole fraction to molar substance. Another disadvantage is that the code might get more verbose as it may require auxiliary variables with an appropriate unit when converting between fraction and substance.
- Advantages are that it will be possible to support and calculate property models written in extensive variables and have better support of partial derivatives from external properties tools.

Currently the C-interface supports both mole fraction and molar substance by having an extra input that defines the unit of the fraction vector.

4.3.3 External object

The external object is a pointer to an instance of a Material class on the C++ side. It consists of:

- A pointer to a property calculator, i.e. an instance of e.g. RefProp or CAPE-OPEN where one calculator instance is shared between external objects with the same calculator key.
- An instance of a cache which may be used by a calculator to extract start values for iterative calculations.

On the Modelica side an external object should be associated with variables from one thermodynamic state set. An advantage with the model based approach is that these details can be hidden from the user which avoids the risk of a user breaking the rule and thereby mess up the caching.

An illustration of the structure is found in **Figure 4**.

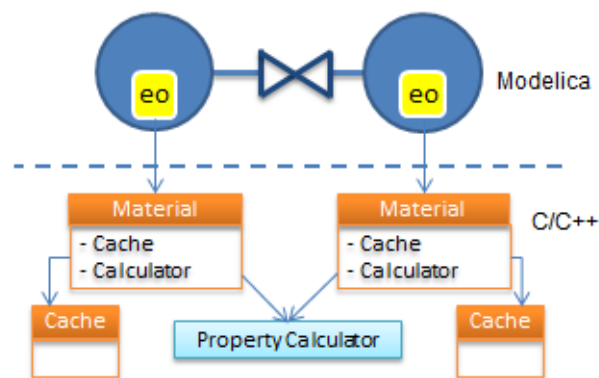


Figure 4. Illustration of the external object structure.

4.4 Challenges with external property code

Most of the available external property packages have not been designed to be used for dynamic simulations. General problems are:

- Error handling when calling properties outside their validity region.
- Limited support for partial derivatives.
- Lack of support to speed up iterative calculation by providing good start values.
- No access to the used tolerances, which may cause numerical problems when creating numerical derivatives.
- Non converging regions.

We have seen in this project that without any additional handling of the validity region issue, simulation will often crash during initialization or simulation. An explanation is that even if a simulation model is set-up to operate within the validity region, the solver might call property routines with invalid inputs when it tries to find a solution for a system of non-linear equations or when it test a large step-size.

In the external interface we decided that it should be the property calculator responsibility to handle this as different property types such as e.g. transport and equation of state based properties may have different validity regions and might be a function of composition.

4.5 CAPE-OPEN

“CAPE-Open standards are the uniform standards for interfacing process modelling software components developed specifically for the design and operation of chemical processes” (Colan, 2015).

The only currently widely adopted standard for thermodynamic property packages is the CAPE-OPEN Thermodynamic and Physical Properties. The backend that has been developed supports both the 1.0 and 1.1 version of the specification.

4.5.1 Disadvantages

Following disadvantages with the CAPE-OPEN thermo interface should be considered (Szczepanski, 2013).

- Missing calculations of: critical properties, phase boundaries, phase stability test.
- No support of flash derivatives (derivatives of flash outputs w.r.t flash specifications with phases in equilibrium)
- Single calculation, in some circumstances it would be useful to calculate properties for an array of inputs

Another disadvantage is that it contains several internal function calls which create an overhead in computation time. And although it was intended as a cross-platform specification, in practice CAPE-OPEN is only supported on Windows.

4.6 RefProp interface

A backend to RefProp has been developed. An early version of the interface was successfully tested on a full air-conditioning cycle model using the single component media R134a. The computational time of the simulation was in the same order as when a corresponding native Modelica implementation of R134a was used. But RefProp does not seem to be suited to be used for larger system simulations for mixtures due to the disadvantages mentioned in chapter 4.4 and that it by default use highly precise multi-parameter equation of state which is rarely used for mixtures due to the computational effort (Schultze, 2014). To overcome these limitations further analysis is needed.

5 Application test case

To verify the overall interface structure a Modelica application library DistillationColumn was created based on work by (Yasaman, 2012). The library has been modified so it is easy to test different continuous state selections and property function inputs.

5.1 Native Modelica Air media

A native Modelica air media was implemented based on work by (Yasaman, 2012). It is a three component model where the phase equilibria conditions are described by the Rachford-Rice equation (Lämås, 2012) using a declarative approach. The equations are solved by the tool's non-linear solver. The vapor phase is described by an ideal gas volumetric equation of state, a linear polynomial for the heat capacity and polynomials adapted to experiment data of the fugacities. The liquid phase uses an incompressible assumption where density and specific heat capacity

are constant and activity coefficients have been adapted to experimental data.

5.2 Air separation column

The lower pressure column in a cryogenic air separation unit was chosen to be used as a test case. Nitrogen and argon is separated from the liquid at atmospheric pressure and a temperature around 85-115K. Liquid with high concentration of oxygen is extracted from the bottom. The column is modeled by 40 equilibrium stages with a total of 164 continuous time states using a 3 component media (nitrogen, oxygen, argon)

5.2.1 Experiment description

Boundary conditions were set to fixed values except for the heat source which increase its value after 100 seconds. Initial transients are present due to the model is not initialized in steady-state

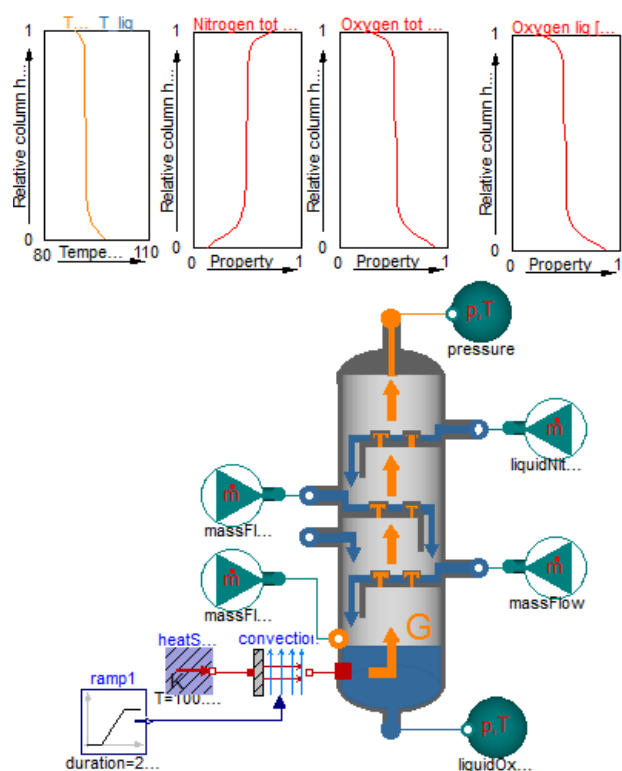


Figure 5 Model of the lower pressure column in a cryogenic air separation unit.

The model was simulated with Dymola 2015 FD01 using the Dassel solver with a relative tolerance of $1e-5$ and a non-equidistant time grid. A standard desktop computer (Intel i7, 8GB Ram and 64-bit Windows operating system) was used for the simulation.

5.2.2 Case 1: Simulation with native Air Modelica media

Result, the simulation finished in 9.46s using 383 successful time steps. The result agreed with result presented in (Yasaman, 2012).

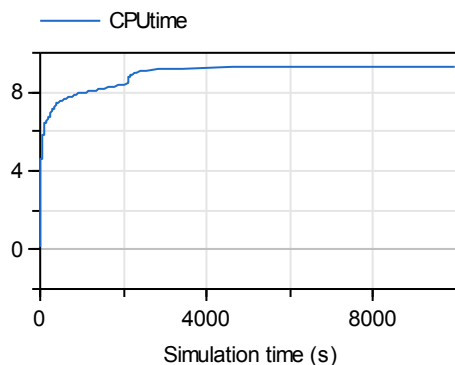


Figure 6. CPU-time with native Modelica media.

The result shows that it is possible to implement an efficient media with the proposed interface using a declarative approach. But good start values are required to succeed with the initialization.

5.2.3 Case 2: Simulation with RefProp Air media

The same model was simulated with a 3 component air media from RefProp. Several different states selection and property input combinations were tested but the tests were unsuccessful due to solver failure when calling properties outside their validity region or due to the solver getting stuck.

Explanation for the slow simulation might be that RefProp uses very high accurate media models that are computational expensive to calculate and that each new flash calculation restarts from scratch every time it is called. For the general flash routines there is no possibility to provide start values. Currently the Modelica-C interface does not support differentiation of all properties which creates numerical Jacobians which are more computational expensive. There might also be other explanations why the simulation performance is not satisfactory. This has to be analyzed further.

6 Limitations

Currently there are restrictions from the used tool and in the Modelica language which makes it harder to use the model based media structure.

6.1 Modelica tools

Following limitations have been observed for different tools:

- Not possible to calculate iteration start values from a property model.
- Not possible to calculate structural parameters from a function using an external object.

The first limitation is severe if a model contains iteration variables that are not equal to a model's start value parameters. If the specific enthalpy is an iteration variable it should be calculated from the given start value parameters as illustrated in Listing 4.

```
parameter SpecificEnthalpy h_start (fixed=false)
annotation(Evaluate=true);
SpecificEnthalpy h(start=h_start);

Medium.MultiPhaseProperties
flash_init(Z=Z_start,p=p_start,T=T_start,
presentPhases=presentPhases,
presentPhasesStatus=presentPhasesStatus,
init(p=p_start, x=fill(Z_start, Medium.nP)),
inputs=MultiPhaseMixture.Interfaces.Inputs.pTX)
initial equation
h_start=flash.h;
```

Listing 4. Calculation of parameters from a model.

The second limitation requires that the user manually specify the number of phases and compounds in the property declaration.

6.2 Modelica specification

Currently it is inconvenient to use a model or block based structure to calculate parameters as illustrated in Listing 4. It would be more user friendly if a model or block could be used in a similar way as a function to calculate parameters.

6.2.1 Solver callback interface

The external interface ExternalMixtureMedia has been designed with a structure that supports caching. The idea is to cache result from a calculation and use it as start values in a next coming calculation to decrease the number of internal iterations and increase robustness. A problem with this approach is that it is not possible to distinguish a function call during normal continuous simulation from one where the steady-state solver desperately tries to find a solution.

During continuous simulation a good strategy would be to use values from the last accepted step. For the steady-state case it might be an idea to let the non-linear solver update the starting values of the iteration variables hidden in these algorithms, when the solver makes good progress.

A solution would be to have the possibility to register a solver callback interface, which could be used to update iteration start values in a controlled way.

A suggestion:

- `onSolverAcceptedStep()` - called by solver/simulation environment when an accepted step has occurred. Place to implement updates of iteration start variables.
- `onSolverSteadyStateProgress()` - called by solver/simulation environment when progress in steady-state solver. Place to update iteration start values variables.

Advantages with introducing callback methods are that the iteration start values can be updated in a controlled way and thereby avoiding the risk of an update during a bad steady-state iteration step.

7 Conclusions

A new framework for thermodynamic properties with support for multi-component multi-phase has been presented. It is the authors hope that this work will initiate a similar development in the process industry as those that have already taken place in the automotive and power industries, where innovative companies have built their innovation processes for systems engineering around the Modelica technology.

The developed Modelica thermo property library use a model based interface which is in line with the Modelica spirit of equation based modelling. The model based interface makes it possible to implement a thermo property model using a declarative approach and the concept was demonstrated by simulating a column in a cryogenic air separation unit.

This work should be seen as a starting point for a model based framework for multi-component multi-phase thermo properties in Modelica. It is possible to improve the framework in following directions:

- Implement an infrastructure for native Modelica implementation of fluids with support of various equations of states and mixing rules including phase equilibrium solvers. The later could be an interesting research topic on how to best formulate these algorithms in a declarative way. A difficulty with the equation based approach is the initialization part, where it would be interesting to see how property models can be formulated to better support initialization. For example by using the homotopy operator.
- Extend the C-interface back end to support more property packages such as MultiFlash.
- Adding additional functionality such as reaction properties.

It would also be interesting to create use cases for the other application mentioned in section 2.1. We

encourage people to take part of continuing the development.

Acknowledgements

The work has been partially funded by the Seventh Framework Programme of the European Union (project MODELICAPROP, Clean Sky number 325975). The financial support from the European Union is highly appreciated.

References

- Ali Baharev and Arnold Neumaier. Chemical Process Modeling in Modelica, *Proceedings of the 9th International Modelica 2012 Conference*, Munich, Germany, September 3-5 2012.
- Stephen Boyd and Lieven VandenBerghe. Convex Optimization, *Cambridge University Press*.
- CAPE OPEN, Thermodynamic and Physical Properties v1.1, May 2011, Downloaded from <http://www.colan.org> (accessed 2015-05-17).
- Francesco Casella and Christoph Richter, ExternalMedia: A Library for Easy Re-Use of External Fluid Property Code in Modelica, *Modelica Conference Proceedings*, 2008.
- Colan, <http://www.colan.org/index-16.html>, accessed 2015-05-17.
- Karin Dietl, Kilian Link and Gerard Schmitz. Thermal Separation Library: Examples of Use, *Proceedings of the 8th International Modelica 2011 Conference*, Dresden, Germany, March 20-22 2011.
- Fluid property library; a common interface to various state-of-the-art thermodynamic and transport property models. <http://www.asimptote.nl/software/fluidprop/> (accessed 2015-05-17).
- Johannes Gernert, Andreas Jäger and Roland Span. Calculation of phase equilibria for multi-component mixtures using highly accurate Helmholtz energy equations of state, *Fluid Phase Equilibria* 375 (2014) 209–218.
- Lemmon, E.W., Huber, M.L., McLinden, M.O. NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP, Version 9.1, *National Institute of Standards and Technology, Standard Reference Data Program*, Gaithersburg, 2013.
- James C. M. Li, Clapeyron Equation for MultiComponent Systems, *The Journal of Chemical Physics* volume 25. number 3 september. 1956.
- Hans Lämås, Algorithms for Multi-component Phase Equilibrium Models in Modelica, *MSc Thesis*, Chalmers University of Technology, Gothenburg, Sweden, 2012.
- Yasaman Mirsadraee. Dynamic modeling and simulation of a cryogenic air separation plant, *Msc Thesis*, Linköping, Sweden, 2012

- J.M. Mollerup and M.L. Michelsen. Calculation of Thermodynamic Equilibrium Properties, *Fluid Phase Equilibria* 74 (1992) 1–15. 1992
- W Morton. Equation oriented simulation and optimization, *Proc Indian Natn Sci Acad* 69, (2003) pp. 317-357. 2003
- Hans Olsson, Hubertus Tummescheit and Hilding Elmqvist. Using Automatic Differentiation for Partial Derivatives of Functions in Modelica, *Proceedings of the 4th International Modelica 2005 Conference*, Hamburg, Germany, March 7-8 2005.
- Vipul Parekh and Paul Mathias, Efficient flash calculations for chemical process design – extension to the Boston-Britt Inside-Out flash algorithm to extreme conditions and new flash types, *Computers and chemical engineering*, vol 22 pp 1371-1380 (1998)
- Modelica Association, Modelica Language Specification, Version 3.3, 2015 <https://www.modelica.org/documents/ModelicaSpec33.pdf>, accessed 2015-05-17.
- Modelica Association Libraries. Available at <https://www.modelica.org/libraries>, accessed 2015-05-17.
- C. Schultze, A Contribution to Numerically Efficient Modelling of Thermodynamic Systems, PhD thesis, Technische Universität Braunschweig, Fakultät für Maschinenbau., (2014)
- Richard Szczepanski, Physical Property Modelling – MultiFlash and CAPE-OPEN. *Presentation for ModelicaProp Workshop 9-10 Oct*, Delft, 2013.
- Mathis Thorade and Ali Saadat, Partial derivatives of thermodynamic state properties for dynamic simulation, *Environ Earth Sci* 70:3497–3503. 2013
- C. Trapp, F.Casella, T. Stelt, P. Colonna. Use of External Fluid property Code in Modelica of a Pre-combustion Co2 Capture Process Involving Multi-Component, Two-Phase Fluids, *Proceedings of the 10th International Modelica 2014 Conference*, Lund, Sweden, March 10-12 2014.
- Hubertus Tummescheit, Jonas Eborn, Chemical Reaction Modeling with Thermofluid/MF and MultiFlash, *Proceedings of the 2nd International Modelica Conference*, Munich, 2002.
- K. Wellner, C. Trapp, G. Schmitz and F.Casella. Interfacing Models for Thermal Separation Processes with Fluid Property Data from External Sources, *Proceedings of the 10th International Modelica 2014 Conference*, Lund, Sweden, March 10-12 2014.
- J.Åkesson, W. Braun, P.Lindholm, B.Bachmann, , Generation of Sparse Jacobians for the Function Mock-Up Interface 2.0, *Proceedings of the 9th International Modelica Conference*, Munich, 2012.

Modeling of the German National Standard for High Pressure Natural Gas Flow Metering in Modelica[®]

von der Heyde, Michael¹ Schmitz, Gerhard² Mickan, Bodo³

¹Institut für Elektrische Energiesysteme und Automation, TUHH, Germany, heyde@tuhh.de

²Institut für Thermofluidodynamik, TUHH, Germany, schmitz@tuhh.de

³Physikalisch-Technische Bundesanstalt, Germany, bodo.mickan@ptb.de

Abstract

The German national metrological institute Physikalisch-Technische Bundesanstalt uses a High Pressure Piston Prover as the primary standard for high pressure natural gas flow meters. The High Pressure Piston Prover measures the gas flow rate using the time a piston needs to displace a defined enclosed volume of gas in a cylinder. Fluctuating piston velocity during measurement can be a significant source of uncertainty if not considered in an appropriate way (Mickan et al., 2010). A computational model written in Modelica[®] was developed to investigate measures for the reduction of this uncertainty. Validation of the model shows good compliance of the piston velocity in the model with measured data for certain volume flow rates. Reduction of the piston weight, variation of the start valve switching time and integration of a flow straightener were found to reduce the piston velocity fluctuations in the model significantly.

Keywords: Modeling of Multi-Domain Physical Systems, Modelica[®], High Pressure Piston Prover, High Pressure Natural Gas Flow Metering

1 Introduction

The German national primary standard for high pressure natural gas flow metering is a High Pressure Piston Prover (HPPP). It is used to calibrate transfer standards for high pressure natural gas flow metering and is traceable to the standards of length and time. The HPPP is described in the references (Schmitz and Aschenbrenner, 1990; Physikalisch Technische Bundesanstalt, 1991; Physikalisch- Technische Bundesanstalt, 2009). It is operated and owned by the German national metrological institute Physikalisch-Technische Bundesanstalt (PTB) and currently installed on the calibration site for gas flow meters pigsarTM in Dorsten, Germany. The calibration facility pigsarTM is also further described in the references (Uhrig et al., 2006; Mickan et al., 2008). Figure 1 shows a picture of the HPPP.



Figure 1. Picture of the High Pressure Piston Prover (Physikalisch- Technische Bundesanstalt, 2009).

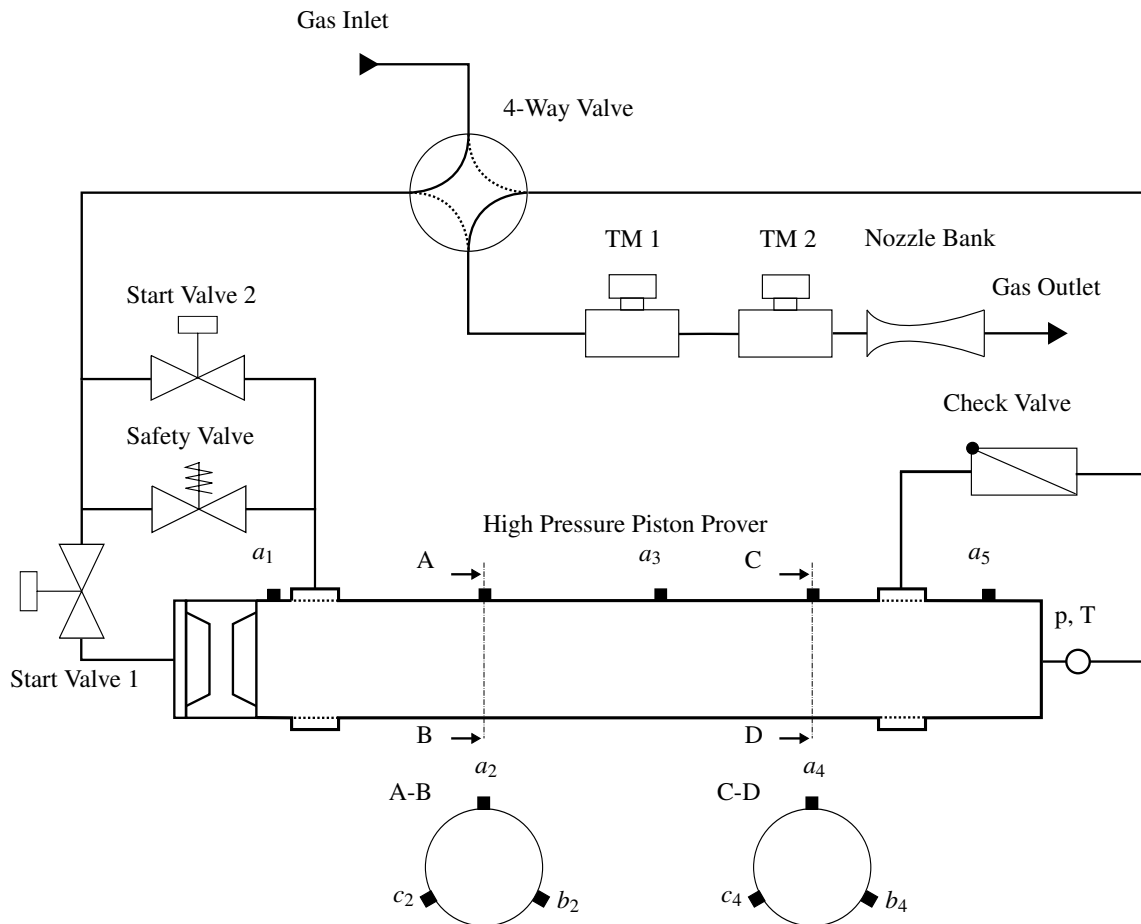
The uncertainty of high pressure natural gas flow meters and therefore also the uncertainty of the HPPP as their primary standard in Germany is of major importance for the trade with natural gas.

2 The Calibration Setup Including the HPPP

The HPPP is the central element of the setup used to calibrate transfer standards. The HPPP consists basically of a piston in a cylinder. Several indicators are mounted on the cylinder to signal the piston position. The pressure and temperature at the HPPP are measured downstream of the cylinder. For the calibration process several other components need to be included in the setup and considered, such as the transfer standards, valves and a nozzle bank.

The HPPP can be operated with inlet pressures up to 90bar and flow rates up to 480m³/h (Physikalisch Technische Bundesanstalt, 1991).

The whole calibration setup including the HPPP is shown in Figure 2.



- a_1, a_5 : Position Indicator
 a_2, b_2, c_2 : Measurement Start Indicator
 a_3 : Half-Way Indicator
 a_4, b_4, c_4 : Measurement Stop Indicator

Figure 2. Scheme of the calibration setup using the High Pressure Piston Prover (Schmitz and Aschenbrenner, 1990).

Start valve 2 is used to initiate the movement of the piston, whereas start valve 1 is needed to prevent movement of the piston in between calibration runs due to the pressure drop across start valve 2. The 4-way valve is needed to revert the gas flow direction and move the piston back to its starting position after each calibration run. A check valve is used to prevent gas from flowing past the piston during the start of the reverse movement and a safety valve is included to prevent high forces on the piston at the end of the piston reverse movement.

Turbine Meters (TM) are used as transfer standards. TM measure the mass flow rate using the rotational speed of a turbine inserted in the fluid flow. The rotational speed of the turbine is metered using magnetically induced discrete signals. Two TM are connected in a row to minimize random measuring errors. The pressure at the TM is measured at their reference point and the temperature 2 diameters downstream of the TM.

The nozzle bank is used to set the flow rate. The critical nozzles are not necessary for the operation of the HPPP but provide the advantage to decouple the calibration setup from pressure fluctuations downstream of the nozzle bank. It consists of several critical flow nozzles in parallel connection. The pressure downstream of the nozzles is always low enough to ensure critical flow in the nozzles.

3 The Calibration Process

The closing of start valve 2 commences the running-in phase. The motion of the piston is indicated by the piston position indicator a_1 . The measurement phase starts as the piston passes indicators a_2, b_2, c_2 and ends as the piston passes the indicators a_4, b_4, c_4 . The volume flow rate is determined as stated in equation 1 from the vol-

ume in between the indicators V_{PP} and the time span $\Delta_{pp}t$ as given by the piston position indicator signals. It is therefore traceable to standards of length and time.

$$\dot{V}_{PP} = \frac{V_{PP}}{\Delta_{pp}t} \quad (1)$$

The signals of the TM are simultaneously counted. The volume flow rate \dot{V}_{TM} can be determined using the relationship between the number of signals per time period indicated by the TM and the volume flow rate, known from previous calibration of the TM.

The calibration result is the relative deviation of the corrected volume flow rate as indicated by the TM \dot{V}_{TM}^c and the corrected volume flow rate as indicated by the HPPP \dot{V}_{PP}^c . The relative deviation f is determined in equation 2. It can be used to correct the TM in further measurements or calibration steps.

$$f = \frac{\dot{V}_{TM}^c - \dot{V}_{PP}^c}{\dot{V}_{PP}^c} \quad (2)$$

Several corrections are used in equation 2 to improve the calibration accuracy. These corrections are explained in the following.

1. The volume flow rate indicated by the turbine meters \dot{V}_{TM} is corrected as shown in equation 3 to prevent an error caused by the discrete nature of the TM signals. $\Delta_{pp}t$ is the duration of the measurement phase as determined from the piston position indicator signals. $\Delta_{TM}t$ is the time span from the first TM signal after the start of the measurement phase to the first TM signal after the end of the measurement phase.

$$\dot{V}_{TM}^c = \dot{V}_{TM} \frac{\Delta_{pp}t}{\Delta_{TM}t} \quad (3)$$

2. The temporal mean density over the measurement phase at the piston prover $\bar{\rho}_{PP}$ and at the TM $\bar{\rho}_{TM}$, both determined from measured pressure and temperature, are used to take the density changes along the gas flow direction into account as shown in the first term of equation 4.

$$\dot{V}_{PP}^c = \dot{V}_{PP} \frac{\bar{\rho}_{PP}}{\bar{\rho}_{TM}} + \frac{V_E}{\Delta_{pp}t} \frac{\rho_S - \rho_E}{\bar{\rho}_{TM}} \quad (4)$$

3. The temporal change of stored mass in between the cylinder and the TM during the measurement phase is taken into account as shown in the second term of equation 4, with V_E being the enclosed volume, ρ_S the spatial mean density in the enclosed volume at the start of the measurement phase and ρ_E the spatial mean density in the enclosed volume at the end of the measurement phase.

4 Uncertainty of the Calibration and Motivation for the Model

Several possible errors in the calibration process lead to the measurement uncertainty of the calibrated TM. These are

1. uncertainty in the determination of the volume in between piston position indicators,
2. uncertainty in the determination of the mean density,
3. repeatability of the TM measurement,
4. leakage between piston and cylinder,
5. dynamic error of the TM,
6. uncertainty in the determination of the stored mass in the enclosed volume.

The dynamic error of the TM is a consequence of the incorrect measurement of fast fluctuating volume flow rates due to turbine inertia. This error can be diminished using a mathematical correction method (Mickan et al., 2010), but the correction method as well leads to uncertainties.

The uncertainty of the calibrated transfer standards is 0.06 % (Physikalisch- Technische Bundesanstalt, 2009; Mickan et al., 2008). The last two listed errors combined lead to an uncertainty of 0.035% (Mickan et al., 2010). They are of dynamic nature and a consequence of piston velocity fluctuations. Figure 3 shows measured data for the piston velocity fluctuations in the measurement phase.

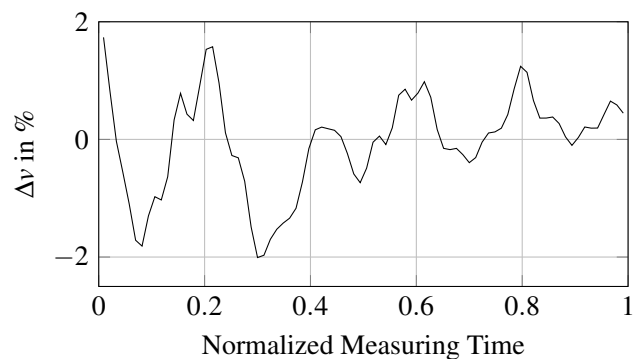


Figure 3. Relative deviation of the measured piston velocity from it's mean velocity Δv over the normalized measuring time.

The developed model is aimed to reproduce these piston velocity fluctuations and to find measures to reduce the fluctuations and therefore the uncertainty of the HPPP.

5 Description of the Model

Modelica[®] was chosen as the language to describe the dynamic and physical system. A graphical representation of the model is shown in Figure 4.

Several general assumptions were used in the model. Those are

1. pressure losses are proportional to the dynamic pressure,
2. the gas flow is one dimensional,
3. the system is adiabatic,
4. potential energy of the gas can be neglected,
5. no leakage occurs,
6. the heat transfer in the gas can be neglected in comparison to convective energy transport.

The gas at the inlet to the HPPP is assumed to have a constant temperature and pressure. This is consistent with data from measurements and is modeled using a supply volume of infinite size from the Modelica Standard Library (MSL). Equation 5 and 6 set these boundary conditions with T_{IN} being the inlet temperature and p_{IN} the inlet pressure.

$$T_{IN} = \text{const.} \quad (5)$$

$$p_{IN} = \text{const.} \quad (6)$$

The nozzle bank sets another boundary condition. It can be modeled as a single nozzle with a larger critical diameter. The used nozzles comply with ISO 9300 (International Organization for Standardization, 2005). The nozzle is modeled using a constant critical volume flow rate \dot{V}_N as shown in equation 7.

$$\dot{V}_N = \text{const.} \quad (7)$$

Equation 8 is used to determine the mass flow through the nozzle \dot{m}_N from the critical volume flow rate \dot{V}_N and the upstream density ρ .

$$\dot{m}_N = \dot{V}_N \rho \quad (8)$$

The valves are taken from the MSL. They have a linear opening function $Y(t)$ and the mass flow \dot{m}_V is proportional to the pressure drop across the valve $\Delta_V p$ as shown in equation 9 with \dot{m}_V^n and $\Delta_V^n p$ being the nominal mass flow and pressure drop.

$$\dot{m}_V = \Delta_V p \frac{\dot{m}_V^n}{\Delta_V^n p} Y(t) \quad (9)$$

The start valves are used in the HPPP model to eliminate the influence of guessed initial conditions on the piston movement, as the stationary flow condition at the beginning of the running-in phase is not known.

The medium in the HPPP is natural gas. Due to the high pressure and high precision of the HPPP a real gas model is necessary. A Modelica Implementation of GERG 2008 with a constant gas composition out of 10 elements is used. GERG 2008 derives the equation of state for natural gas from the free energy. It is described in detail in the references (Kunz et al., 2007; Kunz and Wagner, 2012).

The enclosed gas volumes in the measuring cylinder change with piston movement. They can store mass m and internal energy mu as stated in equation 10 and 11. The volumes have i inlets or outlets. h is the specific enthalpy, v the mean velocity in a cross area A , p the static pressure and V the Volume. The pressure losses at inlets and outlets Δp are considered using constant coefficients ζ_A as shown in equation 12 with $\bar{\rho}$ being the mean density. No gradient for the thermodynamic state and no fluid friction is considered in the volumes.

$$\frac{dm}{dt} = \sum_{i=1}^n \dot{m}_i \quad (10)$$

$$\frac{d(mu)}{dt} = \sum_{i=1}^n \dot{m}_i \left(h_i + \frac{v_i^2}{2} \right) + p\dot{V} \quad (11)$$

$$\Delta p = \zeta_A \frac{\bar{\rho}}{2} v_A^2 \quad (12)$$

The position of the piston is determined from the equation of motion 13 with $F_{F,P}$ being the friction force on the piston, $\Delta_P p$ the pressures difference across the piston, A_P the piston cross area, m_P the piston weight and a_P the piston acceleration.

$$a_P = \begin{cases} 0 & \text{for } |F_{F,P}| \geq |\Delta_{PP} p| A_P \\ \frac{\Delta_P p A_P - F_{F,P}}{m_P} & \text{for } |F_{F,P}| < |\Delta_{PP} p| A_P \end{cases} \quad (13)$$

The friction force on the piston is described as the sum of velocity independent coulomb friction F_C , velocity proportional friction F_{PvP} and stribeck friction F_{Se}^{-kvp} as stated in equation 14.

$$F_F = F_C + F_{PvP} + F_{Se}^{-kvp} \quad (14)$$

The coulomb friction F_C is modeled as a function of the piston position s_P . This function is determined by measuring the power consumption of a linear motor moving the piston slowly through the cylinder. Measured data is only available for 80 % of the cylinder length. After that the coulomb friction is assumed

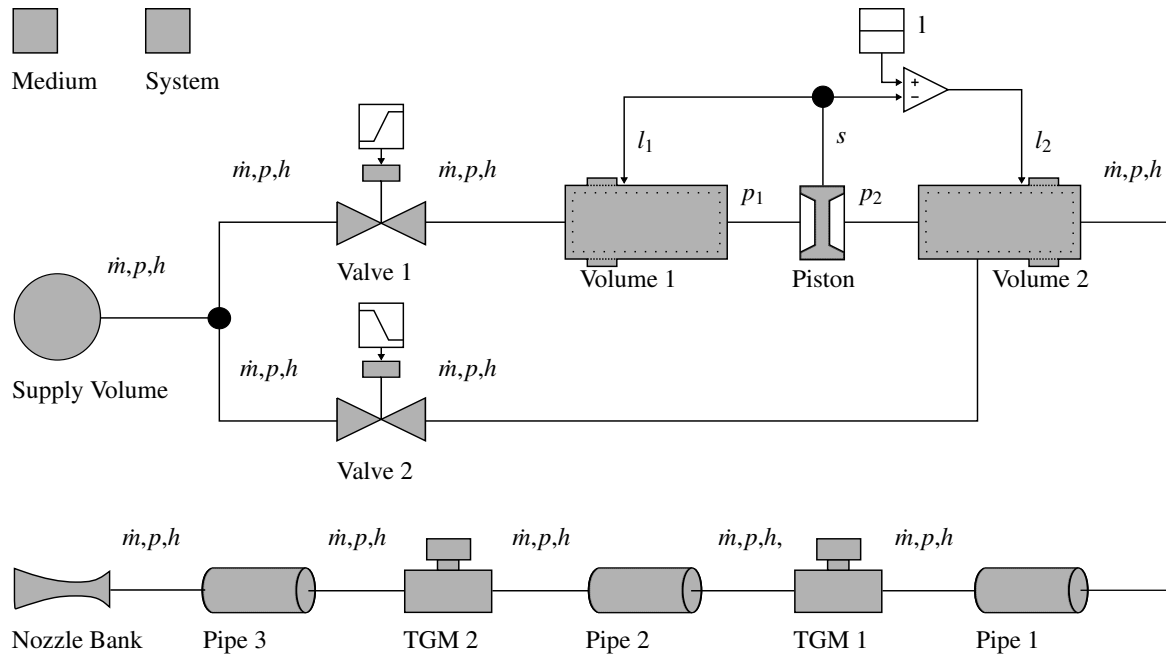


Figure 4. Graphical representation of the computational model.

constant. The measured coulomb friction is shown in Figure 5.

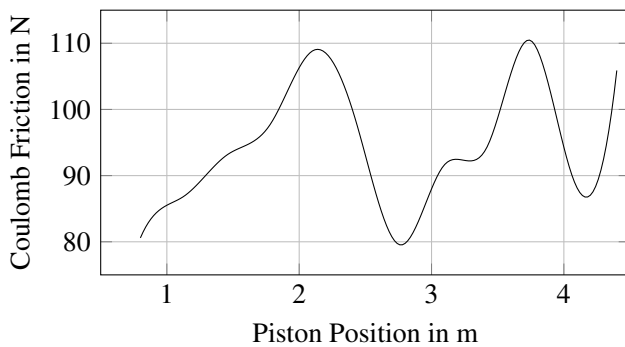


Figure 5. Measured coulomb friction as a function of the piston position.

The velocity proportional friction F_P was determined from measuring the pressure difference across the piston for different piston velocities and by linear interpolation of the measured data points.

The pipes are taken from the MSL. They can store mass m , internal energy mu and momentum mv as stated in equations 15, 16 and 17. A spatial discretisation in the direction of fluid flow is used, leading to a number of finite volumes in the pipe. Each volume reaches from cross area i to cross area $i + 1$. In equation 15, 16 and 17 \dot{m} is the mass flow, h the specific enthalpy, v the mean velocity, A the cross area, p the pressure and F_F the pipe friction force.

$$\frac{dm}{dt} = \dot{m}_i + \dot{m}_{i+1} \quad (15)$$

$$\begin{aligned} \frac{d}{dt}(mu) &= \dot{m}_i h_i + \dot{m}_{i+1} h_{i+1} + \\ &\quad \frac{1}{2}(vA(p_{i+1} - p_i) + vF_F) \end{aligned} \quad (16)$$

$$\frac{d}{dt}(mv) = \dot{m}_i |v_i| + \dot{m}_{i+1} |v_{i+1}| - A(p_{i+1} - p_i) - F_F \quad (17)$$

The turbine meters are modeled using a constant pressure loss coefficient ζ_{TM} as stated in equation 18 with $\Delta_{TM}p$ being the pressure loss, $\bar{\rho}$ the spatial mean density and v_A the mean velocity in the cross area A .

$$\Delta_{TM}p = \zeta_{TM} \frac{\bar{\rho}}{2} v_A^2 \quad (18)$$

The pressure loss coefficient ζ_{TM} is taken from measurements. The relation between the indicated volume flow rate \dot{V}_{TM}^{ind} and the true volume flow rate \dot{V}_{TM} in the TM can be modeled as shown in equation 19 (Mickan et al., 2010). The time constant τ is modeled as stated in equation 20 using a linear relation between the time constant and the initial mass flow \dot{m}_{IC} as approximately found in measurement.

$$\frac{d}{dt}(\dot{V}_{TM}^{ind}) = \frac{\dot{V}_{TM}^{ind} - \dot{V}_{TM}}{\tau} \quad (19)$$

$$\tau = k\dot{m}_{IC} \quad (20)$$

6 Verification of the Model

As measure for the verification and validation the relative deviation of the piston velocity from its mean velocity in the measurement phase Δv is used. Δv represents the piston velocity fluctuations and is calculated as shown in equation 21 using the piston velocity v_P and the mean piston velocity \bar{v}_P determined from the distance Δl and the duration of the measurement phase Δt . For easy comparison of different volume flow rates a normalized time t_n is used in the figures. It is determined in equation 22 from the time t , the start time of the measurement phase t_s and the duration of the measurement phase Δt .

$$\Delta v = \frac{v_P - \bar{v}_P}{\bar{v}_P} \quad \text{with} \quad \bar{v}_P = \frac{\Delta l}{\Delta t} \quad (21)$$

$$t_n = \frac{t - t_s}{\Delta t} \quad (22)$$

For Integration the Solver Dassl included in Dymola[®] is used (Dassault Systemes, 2014) with a relative tolerance of 10^{-6} . Further decrease of the relative tolerance does not change the model trajectory as shown in Figure 6. No major change in the trajectories is detected when using other high order variable step solvers implemented in Dymola[®].

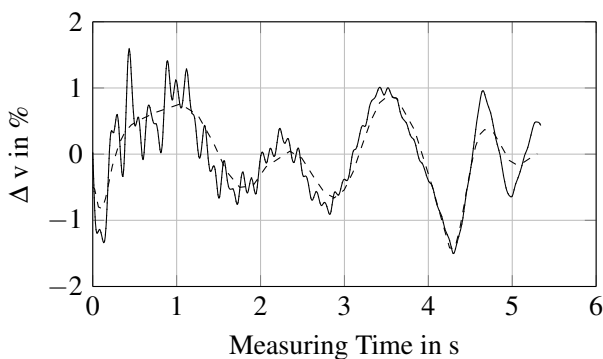


Figure 6. Relative deviation of the piston velocity from the mean velocity Δv in the model for different relative solver tolerances using Dassl.
 - - - - $TOL = 10^{-4}$ — $TOL = 10^{-6}$ - · - $TOL = 10^{-8}$

Due to calculation time it is not functional to use a high number of finite pipe volumes in conjunction with real gas behavior. Here 4 discrete volumes in the first pipe and 2 volumes in the 2nd and 3rd pipe are used.

The verification of the model shows increasing frequency and decreasing deflection of the relative piston velocity deviation for increasing inlet pressures as shown in Figure 7.

Due to a shorter duration of the running-in phase, the piston velocity fluctuation resulting from piston

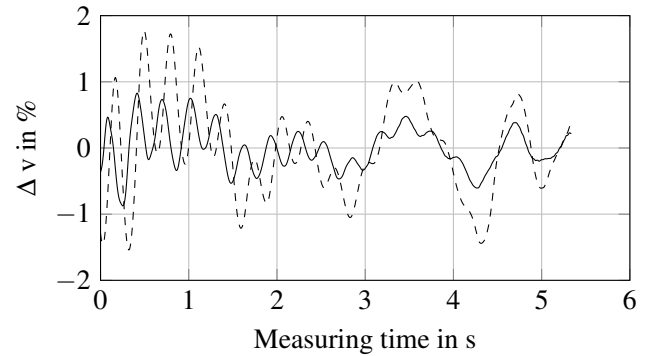


Figure 7. Relative deviation of the piston velocity from the mean velocity Δv in the model over the measuring time for different inlet pressures.
 — $p_{IN} = 50 \text{ bar}$ - - - - $p_{IN} = 20 \text{ bar}$

acceleration remains active during the measuring phase for higher volume flow rates as shown in Figure 8.

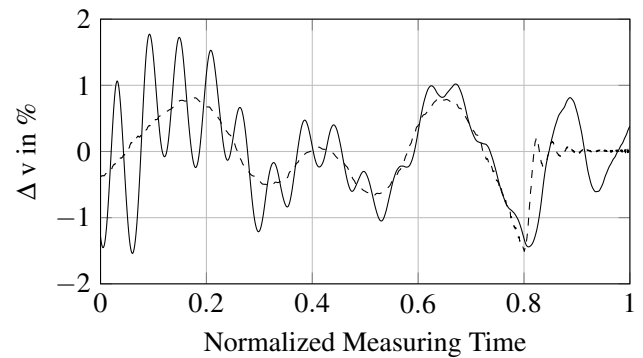


Figure 8. Relative deviation of the piston velocity from the mean velocity Δv in the model over the measuring time for different volume flow rates.
 — $\dot{V}_N = 100 \text{ m}^3/\text{h}$ - - - - $\dot{V}_N = 25 \text{ m}^3/\text{h}$

7 Validation of the Model

The model accuracy is highly relevant due to the low measuring uncertainty of the High Pressure Piston Prover. It depends on the uncertainty of the measured parameters used for the calibration of the model, the mentioned general assumptions, the simplified mathematical description and the accuracy of the numerical algorithm.

Measured data for the piston velocity is used to validate the model. The piston velocity was measured for volume flow rates up to $100 \text{ m}^3/\text{h}$ using a laser distance measurement system.

The model validation shows relatively good accordance of the piston velocity fluctuations with measurement data for a volume flow rate of $100 \text{ m}^3/\text{h}$, as

shown in Figure 9. The ground oscillation as well as the superimposed high frequency oscillation show similar characteristics for a volume flow rate of $100\text{m}^3/\text{h}$. This is also valid for different inlet pressures.

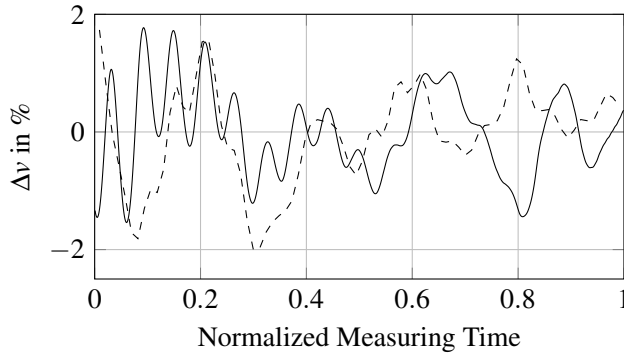


Figure 9. Comparison of the relative piston velocity deviation Δv over the normalized measuring time in the model and in measured data for a volume flow rate of $100\text{m}^3/\text{h}$ and an inlet pressure of 20 bar.

— Simulation - - - - Measurement

For lower volume flow rates the model is not able to reproduce the measured piston velocity fluctuations. As an example the piston velocity fluctuation in the model is compared with measurement data for a volume flow rate of $25\text{m}^3/\text{h}$ in Figure 10. It can be seen, that the high frequency fluctuation in the measured data is not present in the model for a volume flow rate of $25\text{m}^3/\text{h}$.

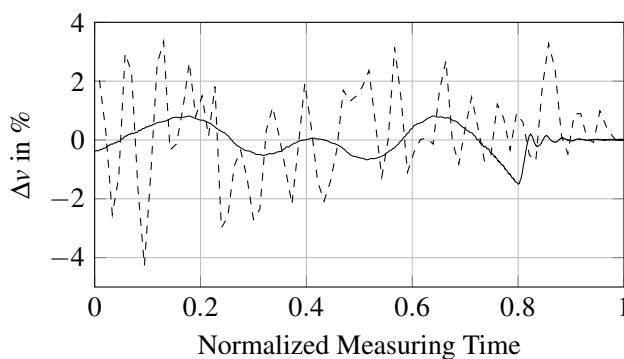


Figure 10. Comparison of the relative piston velocity deviation Δv over the normalized measuring time in the model and in measured data for a volume flow rate of $25\text{m}^3/\text{h}$ and an inlet pressure of 20 bar.

— Simulation - - - - Measurement

Physical effects that are not included in the model and measurement uncertainties, both during the determination of the parameters for calibration and of the data for validation, might play an important role for low volume flow rates.

8 Constructional Means for Piston Velocity Fluctuation Reduction

The model is used to evaluate three different ways to reduce the piston velocity fluctuations in the measuring phase. The maximum deviation of the piston velocity from its mean velocity $\Delta_{\max}v$ is used as a measure for the piston velocity fluctuations. The mean piston velocity deviation would not be an adequate measure here, as it does not limit the important piston velocity deviation at the start and end of the measurement phase, whereas the maximum piston velocity deviation does.

Accordingly to verification and validation the Solver Dassl included in Dymola[®] (Dassault Systemes, 2014) with a relative tolerance of 10^{-6} and 4 discrete volumes in the first pipe as well as 2 volumes in the 2nd and 3rd pipe are used.

Here the results for a volume flow rate of $100\text{m}^3/\text{h}$ and an inlet pressure of 20 bar are shown.

Figure 11 shows the maximum relative deviation of the piston velocity from its mean velocity in the measuring phase for different piston weights. A lower piston weight leads to lower maximum piston velocity fluctuation in the model. The real piston weight is 21,7 kg. Reducing the piston weight by 50 % would lead to a significant drop of the piston velocity fluctuations. A way to achieve this reduction can be a change of the piston material from aluminum to fiber reinforced polymers.

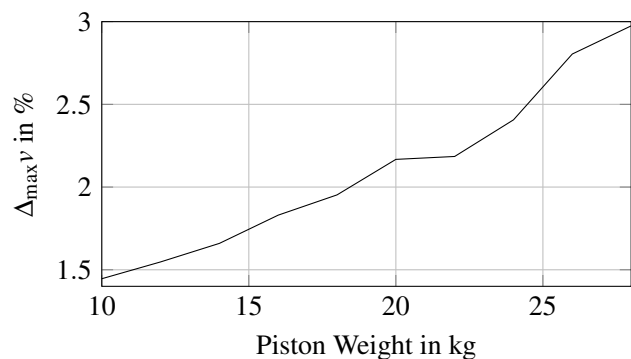


Figure 11. Maximum deviation of piston velocity from mean velocity in measuring phase for different piston weights.

Another way to reduce the piston velocity fluctuations in the model is shown in Figure 12. As can be seen, the switching time of start valve 1 has a strong influence on the maximum deviation of the piston velocity from its mean velocity during the measuring phase below a switching time of 0,4s. The switching time would have to be adopted for other volume flow rates using a controller.

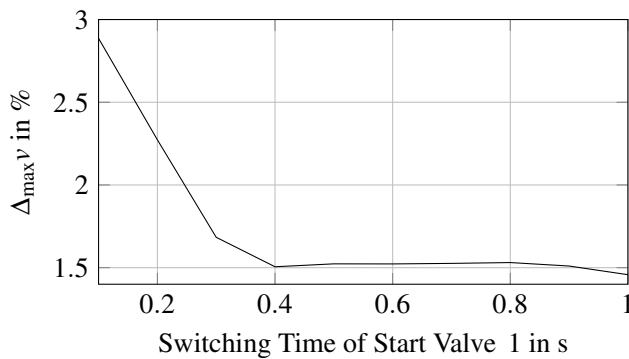


Figure 12. Maximum deviation of piston velocity from mean velocity in measuring phase for different switching times of start valve 1.

Figure 13 shows the maximum relative deviation of the piston velocity from its mean velocity in the measuring phase as a function of the pressure loss coefficient of pipe 1. A significant reduction of the piston velocity fluctuations can be achieved for higher pressure loss coefficients. A possibility to raise the pressure loss coefficient would be the integration of a flow straightener. The error due to a higher pressure loss between the HPPP and the TM could be avoided using correction step 2 as described in section 3.

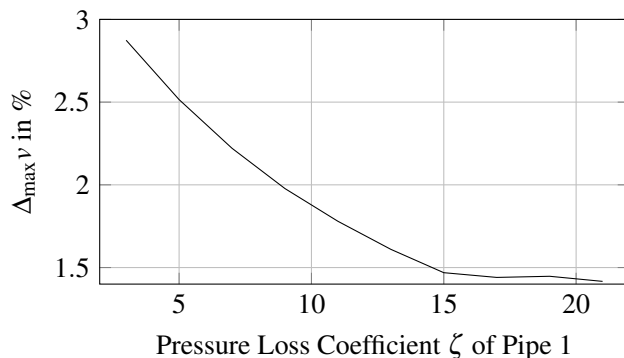


Figure 13. Maximum deviation of piston velocity from mean velocity in measuring phase for different pressure loss coefficients in pipe 1.

9 Conclusions

Modelica[®] proved the adequate language for the modeling of the High Pressure Piston Prover. It was possible to use several components from the Modelica Standard Library containing equations from various physical domains, such as tribology, thermodynamics and fluid flow. Parts of the model are replaceable and reusable due to Modelica[®] being object-oriented. The structure of the model follows the physical structure of the High

Pressure Piston Prover closely. As a consequence of the Modelica[®] acausality the physical equations in the model are comprehensible and errors are easier located during the modeling process.

Three independent ways to reduce piston velocity fluctuations were demonstrated using the developed model. A significant reduction of the maximum piston velocity fluctuation during the measuring phase was found achievable by lowering the piston weight, an appropriate setting of the start valve switching time and the integration of a flow straightener. These measures are expected to reduce the High Pressure Piston Prover measuring uncertainty and can be realized with low effort.

References

- Dassault Systemes. Dymola Dynamic Modeling Laboratory, 2014.
- International Organization for Standardization. ISO 9300, 2005.
- O. Kunz and W. Wagner. The GERG 2008 wide range equation of state for natural gases and other mixtures. *Journal of Chemical Engineering*, 2012.
- O. Kunz, R. Klimeck, W. Wagner, and M. Jaeschke. The GERG 2004 wide range equation of state for natural gases and other mixtures. *GERG Technical Monograph*, 15, 2007.
- B. Mickan, R. Kramer, H. Müller, V. Strunck, D. Vieth, and H.-M. Hinze. Highest precision for gas meter calibration worldwide: The high pressure gas calibration facility pigsar with optimized uncertainty. In *International Gas Union Research Conference*, 2008.
- B. Mickan, R. Kramer, V. Strunck, and T. Dietz. Transient response of turbine flow meters during the application at a high pressure piston prover. In *15th Flow Measurement Conference (FLOMEKO)*, 2010.
- Physikalisch- Technische Bundesanstalt. PTB mitteilungen, special issue volume 119 no.1, 2009.
- Physikalisch Technische Bundesanstalt. Prüfschein der Rohrprüfstrecke, 1991.
- G. Schmitz and A. Aschenbrenner. Experience with a piston prover as the new primary standard of the federal republic of germany in high pressure gas metering, 1990.
- M. Uhrig, P. Schley, M. Jaeschke, D. Vieth, K. Altfeld, and I. Krajcin. High precision measurement and calibration technology as a basis for correct gas billing. In *23rd World Gas Conference*, 2006.

Automatic Regression Testing of Simulation Models and Concept for Simulation of Connected FMUs in PySimulator

Adeel Asghar¹ Andreas Pfeiffer² Arunkumar Palanisamy¹ Alachew Mengist¹
Martin Sjölund¹ Adrian Pop¹ Peter Fritzson¹

¹PELAB – Programming Environment Lab, Dept. Computer Science, Linköping University, Sweden,
{adeel.asghar, arunkumar.palanisamy, alachew.mengist,
martin.sjolund, adrian.pop, peter.fritzson}@liu.se

²DLR Institute of System Dynamics and Control, 82234 Weßling, Germany, andreas.pfeiffer@dlr.de

Abstract

The Modelica and FMI tool ecosystem is growing each year with new tools and methods becoming available. The open Modelica standard promises portability but it is important to ensure that a certain model behaves the same in different Modelica tools or in a different version of the same tool. It is also very important (for model evolution) to check that a new version of the same model produces comparable results. Finally, it is desirable to verify that a model exported in FMU form from a Modelica tool gives exactly the same results as the original model. This paper presents a framework for automatic regression testing as part of PySimulator which provides an efficient and concise way of testing if a model or a range of models behaves in the same way in several tools or versions of a tool by checking that the results produced are essentially identical.

The FMI standard has been adopted by many tool vendors and is growing in popularity each year. This paper proposes a concept for building and simulating a system made from connected FMUs generated by different tools. The FMUs for Co-Simulation can be connected together using a GUI. The system model built graphically in this way can be saved for later use or simulated directly inside PySimulator. Active development is going on to support simulation of connected FMUs for Model Exchange.

Keywords: *PySimulator, Regression Testing, Connected FMUs, Parallel Simulation, Wolfram Simulator plugin*

1 Introduction

Due to the success of Modelica and FMI many different tools support these open standards (e.g., see the table of Modelica tools on www.modelica.org/tools and FMI tools on www.fmi-standard.org/tools). To ensure a high quality of models, tools, and their interoperability, it will become increasingly important to have tools available for automatic testing of models with different Modelica / FMI tools. As a first step, the Modelica Association has financed the development of

a CSV comparison tool (ITI, 2013). Currently a tool to test the examples of the Modelica Standard Library is being developed within the Modelica Association (Otter, 2015).

Some Modelica tool vendors have their own features to test models, but only by using their own tool (e.g. OpenModelica or Dymola). What is currently missing is a platform to perform regression testing among different tools. The open source environment PySimulator (Pfeiffer et al, 2012), see also www.pysimulator.org, has the potential to contribute to such a platform because it already supports several different simulator tools and result file formats.

PySimulator is an environment implemented in Python that provides a graphical user interface for simulating different model types (currently Functional Mockup Units, Modelica models, and SimulationX models), plotting result variables and applying simulation result analysis tools. The modularity concept of PySimulator enables easy development of further plugins for both simulation and analysis.

In Section 2 of the paper we have extended the list of simulator plugins for PySimulator by implementing a plugin for Wolfram's SystemModeler. In Section 3 we present the analysis plugin, *testing*, for PySimulator that enables different features necessary to provide convenient regression testing with good performance. In Section 4 we introduce functionalities like automatic simulation of models given by a list in a text file as well as parallel simulation and regression analysis to considerably speed up the computation time on multi-core machines.

As PySimulator is aimed at playing the role of an integration platform, the support of connected FMUs is a further topic of this paper. It is an important feature to run simulations of connected FMUs from different suppliers since the suppliers can protect their knowledge within the FMU and a whole system consisting of several components (represented by FMUs) can be simulated. In Section 5 a concept is introduced on how to describe and simulate connected FMUs within PySimulator.

2 Simulator Plugin for Wolfram SystemModeler

PySimulator supports simulation of models in FMU form or using different Modelica tools via extension plugins. From previous work simulator plugins for tools such as Dymola, SimulationX, and OpenModelica (Ganeson et al, 2012) are available. This section presents a new simulator plugin developed for Wolfram SystemModeler.

Using the existing plugin interface for simulator plugins in PySimulator a new simulator plugin has been implemented: the *Wolfram plugin*. It enables PySimulator to load and numerically simulate Modelica models using Wolfram SystemModeler (Wolfram SystemModeler, 2015).

The Wolfram plugin is integrated into PySimulator via MathLink (Wolfram SystemModeler, 2015) and Pythonica (Edwards, 2012) which connects to Mathematica (Wolfram Mathematica, 2015) and SystemModeler. We used the Wolfram SystemModeler API to support loading a Modelica model, simulating it, and reading the simulation setting file (.sim) which is an XML file to build the variable tree in the variables browser of PySimulator. The overall communication setup with SystemModeler is given in Figure 1.

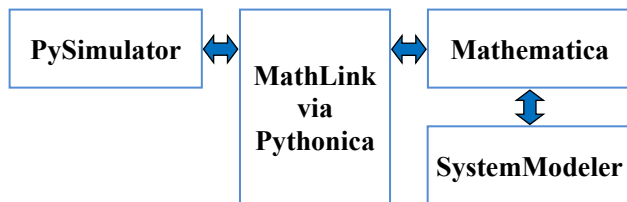


Figure 1. Communication setup with SystemModeler.

All the simulator plugins of PySimulator are controlled by the same Integrator Control GUI. The Wolfram SystemModeler simulator supports five different numerical integration methods (DASSL, CVODES, Euler, RungeKutta, and Heun), all the simulation menu options are supported (error tolerance, fixed step size, etc.).

The start and stop time for the integration algorithm can be changed and one of the integration algorithms can be selected. Depending on the integration algorithms the user can change the error tolerance or the fixed step size before running the simulation.

It is also possible to simulate the list of models using the Wolfram plugin, see Figure 9 in Section 4. The existing PySimulator interface automatically includes the new plugin to the simulators list for simulating a list of models, see also Section 4.1.

3 Regression Testing – Design and Appearance

In this paper, regression testing means the automated simulation of models and the automated comparison of the simulation results with some kind of baseline results (normally also automatically simulated). An automatically generated summary report gives the overview of the whole test results.

Possible applications of such test procedures are the following (Pfeiffer et al, 2013):

- Different versions of a model exist and they are compared to the original version of the model within one tool (model evolution and validation).
- A Modelica model is simulated by different tools and the results are compared to a reference solution (tool validation).
- A Modelica model and its corresponding FMU exported by a tool are compared to each other (FMU export model validation).
- An FMU is exported by different tools for the same model. The results of the FMUs are compared to each other (FMU export tool validation).

The applications are described for one model but they can also be applied to a list of models, e.g., all example models of a Modelica library.

Several parts are necessary to realize the mentioned features within PySimulator:

- Enable the automatic simulation of a given list of models by a defined list of simulator plugins, see Section 4.1.
- Compare the variables of simulation result files with different simulation result formats like Dymola’s mat-format, CSV-format, MTSF-format (Pfeiffer, Bausch-Gall et al, 2012).
- Compute a numerical measure for the deviation of two time-dependent signals.
- Enable automatic walk through result file directories and find result files that can be compared.
- Generate HTML-reports that document the outcome of comparing the variables in the result files.

3.1 Comparing Variables in Result Files

The concept of how to compare the results of model simulations is mainly based on the comparison of two result files. In PySimulator several plugins for different simulation result file formats have been created by the previous work of several contributors¹, see Figure 2.

¹ A. Pfeiffer, M. Otter (DLR), I. Bausch-Gall (Bausch-Gall GmbH), T. Beutlich (ITI GmbH)

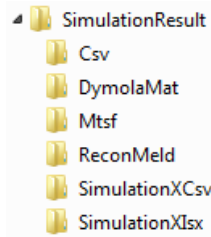


Figure 2. Different result file formats.

These plugins are used to read (and partly write) the simulation result files after the simulation run. Internally in PySimulator the data of the result files is structured according to time series. The concept of time series is in the style of the MTSF format, see (Pfeiffer, Bausch-Gall et al, 2012) for details. All variables based on the same time grid are grouped into a time series. Typically, three types of time series can be found in result files:

- Parameters and constants (a special time series without a time grid),
- Discrete variables (time grid is according to events),
- Continuous variables (time grid is given by the output points of the integrator and by events).

In the current implementation the basic algorithm to compute the deviation between two time dependent signals / variables $x(t)$ and $y(t)$ relies on the following measure:

$$d(x, y) := \frac{\varphi(x - y)}{1 + \varphi(x) + \varphi(y)}$$

with $\varphi(z) := \frac{1}{t_e - t_0} \int_{t_0}^{t_e} |z(t)| dt.$

The deviation measure d can be understood as a combination of the absolute and relative integral error between the two signals x and y on the time interval $[t_0, t_e]$. Due to adding 1 to $\varphi(x) + \varphi(y)$, the denominator is always greater than zero. The inequalities $0 \leq d \leq 1$ hold because of the triangle inequality $\varphi(x - y) \leq \varphi(x) + \varphi(y)$. For constant signals x, y (like parameters or constants of models) we have

$$d(x, y) = \frac{|x - y|}{1 + |x| + |y|}.$$

E.g. for $x = 2$ and $y = 2.01$ we get $d \approx 2e-3$ which is in the order of magnitude of the relative error $0.01/2 = 5e-3$.

To get time dependent functions for the signals of a simulation result file the result points are linearly interpolated. The integrals of piecewise linear functions can easily be computed by an analytic approach – also including discontinuities introduced by events during numerical integration. The main parts of the algorithm and of the computation time is concerned with the (possibly different) time grids of x and y .

Therefore the time series concept fits very well into the algorithm. For each time series only one time grid is defined and the corresponding computational effort for the grid is only done once. On the other hand the time series concept enables reduction of the simulation result file size because only result points are saved when possible changes in the variable can be expected. Because there is no best way to compare signals, the implemented algorithm can easily be exchanged by another (user-defined) algorithm – if necessary.

It is clear that linear interpolation of the result points introduces an error between the linear interpolation and the numerical solution normally available with (much) higher precision. The error of linear interpolation is $O(\Delta t^2)$ with the time grid width Δt , whereas for a numerical integration method e.g. of order 4 the global error between the analytical solution and the numerical approximation is $O(h^4)$ for the time step size h . This means that it does not make sense to compare results accurately computed by high order integration algorithms and finally to compare them on different (wide spaced) time grids with linear interpolation between. Consequently, it is highly recommended to generate equal time grids for the result files to be compared using the dense output functionality by novel integration algorithms.

The concept to define a measure has the advantage that really a number is computed for the deviation between two signals. The alternative approach to only check, if two signals are identical within a given error tolerance gives a true / false information but does not specify how far the signals are away to be within the tolerance. Of course, the deviation number can also be used to check if it is below the error tolerance.

For the user of PySimulator and the testing plugin a GUI has been developed to define regression tests, see Figure 3.

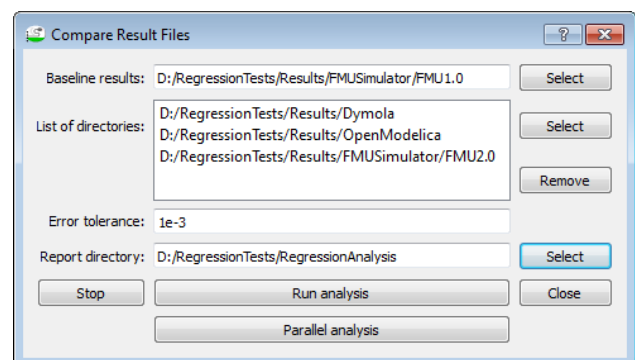


Figure 3. Compare result files GUI.

In the baseline result directory there are result files that are used as a reference to be compared to the result files in the given list of result directories. Each directory is searched for a result file with the same name as the baseline result file (without file suffixes). If there are files with the same names except the file suffix, then these files will be compared using the

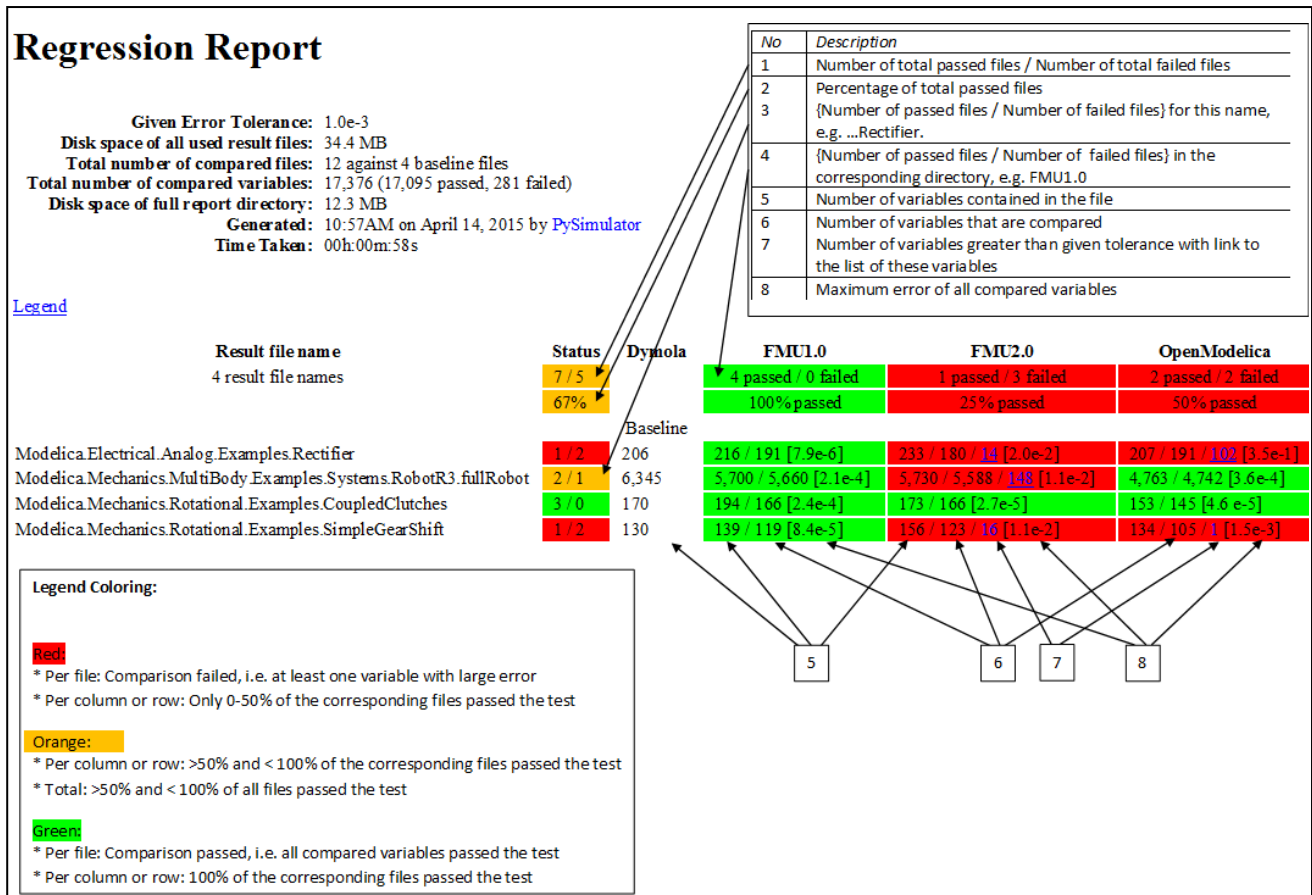


Figure 4. The HTML report for regression analysis.

algorithm described above. Before starting the analysis, the user has to specify an error tolerance up to what deviations between signals are acceptable. The regression report and all corresponding files are generated into the report directory to be defined by the user.

3.2 HTML Report for Regression Testing

The result of the regression testing is a generated HTML report which presents the results of the analysis in a compact and concise way. We have been iterating over the appearance of the HTML report in order to make it more clear and compact while providing enough information to the user about the regression analysis.

The appearance of the current version of the HTML report is given in Figure 4. It includes a table with the given models for simulation and the results obtained by running the given tools. The top left corner gives general information about the regression analysis such as tolerance, used disk space, how many files and signals were compared, generation time, etc.

The legend which gives the meaning of the colors is given below the table with the results and linked from above so that more useful information is displayed close to the top.

The table gives information about the regression testing including: how many comparisons passed or failed, the largest difference between the signals, and the total number of signals in the reference file and in the file generated via simulation. An overview column called “Status” is also present to quickly spot the problematic tests.

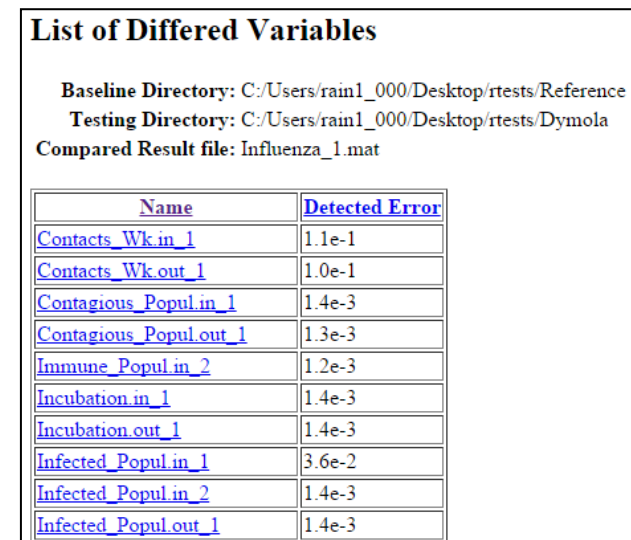


Figure 5. HTML view with all the signals that differ.

The columns in the right part of the report table show as a link how many signals differ with respect to the given tolerance. For example, there are 14 variables for the Rectifier FMU (FMI 2.0) that differ from the baseline simulation by Dymola. One can click on that number and another HTML page will be presented with an overview of the differences (Figure 5). On this page one can see a table containing all the signals that differ, sorted either by the variable name or by the error between signals. To switch between the sorted pages, one can click on the column headers of the table namely “Name” and “Detected Error” to navigate to the respective sorted page.

In this view the user can click on the variable names and a new interactive page is displayed with more information about the difference in the signals.

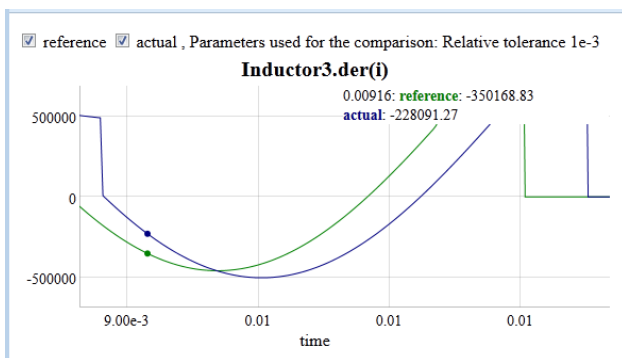


Figure 6. Interactive HTML view with the difference between signals.

In the interactive signal difference HTML view (Figure 6) the user can zoom in and see the actual difference between the selected variables.

3.3 Speed-up of Regression Testing

For many models or models with long simulation times and / or large result files, the task to run the whole regression testing analysis may take a long time. To improve the performance two kinds of parallelization techniques are applied:

- Simulate different models in parallel,
- Compare different result files in parallel.

The simulation of different models is presented in Section 4.1 and the benefits of parallelization in Section 4.2. The comparison of different result files in parallel and the speed-up achieved versus serial comparison is given in Section 4.3.

4 Performance of Regression Testing

In this section we detail the functionality available to simulate models and to perform the regression analysis. The performance improvements gained when parallelization is applied are also presented.

4.1 Automatic Simulation in Batch Mode

In the initial design (Pfeiffer et al, 2012) of simulator plugins in PySimulator the main interface to run a numerical integration of a model was to click and edit through the Integrator Control GUI. This is convenient when experimenting with a few models and the according result files. However, if we want to simulate several models to generate result files (as needed for regression testing), the original procedure will get tedious and error-prone.

For this case we introduced a text file based interface for PySimulator to specify the simulation parameters of a list of models. The format of the text file is rather simple. Currently, data for nine columns has to be inserted for each model to be simulated. Comment lines beginning with # can also be put in the file. The user has to specify:

- The file name (possibly with full path name) of the model or the library,
- The unique model name inside the library,
- An optional name of a sub-directory, where the result file has to be saved,
- The start and stop time of the integration,
- The error tolerance or the fixed step size (depending on the default integration algorithm),
- The number of output intervals for the result file,
- True or false, if result points at events shall be included in the result file.

An example how a simulation setup file looks like is given in Figure 7.

The setup file can easily be generated by some other tools. A prototype is implemented in a scripting function in Dymola to generate the setup file for all models of a Modelica library with an “experiment” annotation.

The setup file can be loaded using the PySimulator GUI interface. An example of how to start the GUI and load the setup file is given in Figure 8.

```
# Setup file for simulation of several models by PySimulator
# Columns to be filled:
# modelFile modelName subDir tStart tStop tol stepSize nIntervals includeEvents
# List of models to be simulated:
"D:/BouncingBall.mo" BouncingBall "" 0.0 2.0 1e-6 10 500 true
"D:/Rectifier.mo" Rectifier "" 0.0 0.1 1e-6 10 500 true
"D:/Rectifier_10.fmu" Rectifier "FMU1.0" 0.0 0.1 1e-6 10 500 true
"D:/Rectifier 20.fmu" Rectifier "FMU2.0" 0.0 0.1 1e-6 10 500 true
```

Figure 7. Content of the simulation setup file Setup.txt.

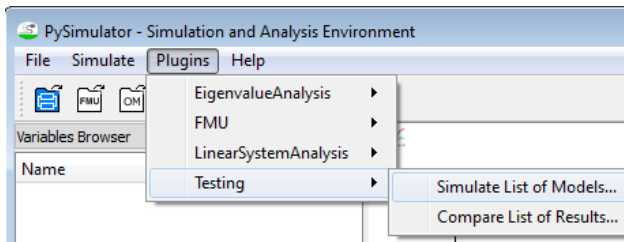


Figure 8. Starting the simulation list of models from GUI.

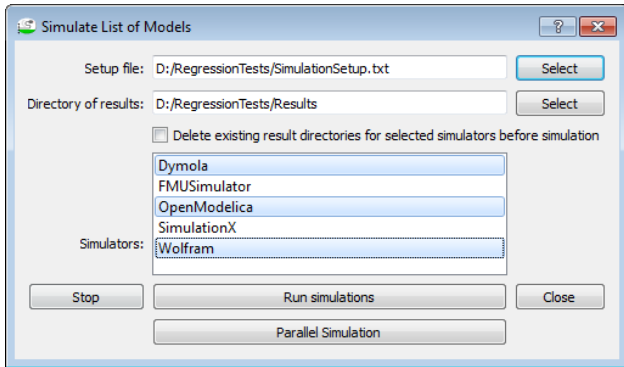


Figure 9. GUI to load a setup file and select the simulator tools from the list.

After selecting “Simulate List of Models...” from the menu, the GUI interface pops up as shown in Figure 9. After loading the setup file the user can select several simulator plugins that shall run the models specified in the setup file. The simulator plugins are able to recognize if they can simulate all model types given in the setup file. Models that cannot be processed are just ignored. Currently, all simulator plugins for Modelica models ignore FMUs and the FMU Simulator ignores Modelica models. The parallel simulation of the models to speed up the whole simulation process is explained in the following section.

4.2 Parallel Simulation

The parallel simulation approach allows the user to simulate models in parallel in different processes, using as many cores as the machine has available, resulting in improved performance. Each model in the list is simulated in a separate directory in order to avoid conflicts that would occur if models use the same file names. Generating the files with the same name can occur due to simulating the same model multiple times in the same project, or due to the simulator using the same name for all models (e.g. dsin.txt, output.log in Dymola).

The Python Multiprocessing Library (Python, 2015) was used to implement the parallelization of simulation runs. Multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency, effectively side-stepping the global Python interpreter lock by using sub-processes instead of threads. Due to this the multiprocessing module allows the programmer to

fully leverage multiple processors on a given machine. The library provides the cross-platform support and is compatible with both UNIX / Linux and Windows operating systems.

We measured the performance of parallel simulation against serial simulation. The list of models is taken from the example models in the Modelica Standard Library 3.2.1 (Modelica Association, 2013). The tests have been performed with the following system configuration:

OS: Windows 8, 64 bit
 Processor: 4-core CPU @ 2.20 GHZ
 RAM: 8 GB

A selection of measurements is listed in Table 1.

Table 1. List of measurements between serial and parallel simulation using the OpenModelica simulator.

<i>Models</i>	<i>Serial [s]</i>	<i>Parallel [s]</i>	<i>Speed-up factor</i>
10	134.9	35.5	3.80
26	349.3	84.1	4.15
52	648.1	195.6	3.31
100	1279.3	381.8	3.35

The table shows that parallel simulation is roughly three to four times faster than serial simulation. If the number of processor cores in the system increases, the speed-up will increase accordingly, as long as there is no shared global memory or disk bottleneck.

4.3 Parallel Regression Analysis

The regression testing as shown in Section 3 is parallelized in the same way as described in the previous section for the simulation runs. The comparison of two result files including loading the files is run in parallel for several result file pairs.

We measured the performance of serial regression testing when compared with the parallel implementation. The tests are performed with the same system configuration as specified in Section 4.2. A selection of measurements is listed in Table 2.

Table 2. List of measurements between serial and parallel regression testing.

<i>Total size of files [MB]</i>	<i>Files compared</i>	<i>Total variables compared</i>	<i>Serial [s]</i>	<i>Parallel [s]</i>	<i>Speed-up factor</i>
1.2	20	387	9	4	2.25
2.4	45	872	19	8	2.37
17.6	100	11206	52	20	2.60
30.0	200	24164	90	27	3.33
47.6	325	36347	178	55	3.23

From the above measurements the parallel regression testing is roughly two to three times faster than serial regression testing.

5 Simulation of Connected FMUs

It is often required to simulate a model containing several FMUs connected to each other. The FMU simulator plugin of PySimulator so far has relied on FMI 1.0 for Model Exchange. As preparation work to support connected FMUs (FMI 2.0), we extended the plugin to cover the FMI standard in version 2.0 (Modelica Association, 2014a) for Model Exchange and for Co-Simulation of a single FMU. Further, we have developed a new simulator plugin which allows connection and simulation of several FMUs. Some details are shown in this section.

5.1 Connections between FMUs

The information about how several FMUs are connected is stored in an XML file. It contains the details about the FMUs and their respective connections required for the simulation. This makes it possible to write the XML file manually and open it in PySimulator.

We have also designed a connection GUI shown in Figure 10 which allows the user to select FMUs and make connections between them. The information is saved into the XML file and can be used again in later sessions.

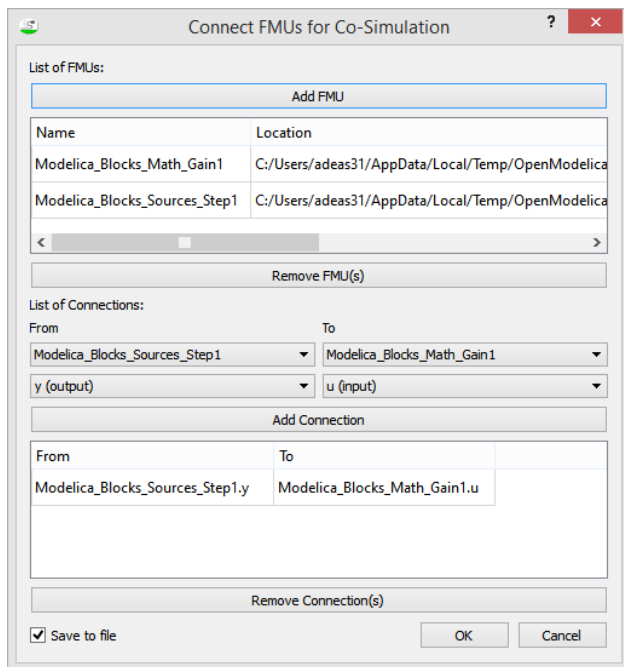


Figure 10. Graphical user interface to connect FMUs.

The according XML schema in Figure 11 contains two main sections namely `fmus` and `connections`. Each `fmu` has a unique name, which is also used as instance name in the simulator, and a `path` to define where the FMU is stored. Each `connection` contains:

- `fromFmuName`: the instance name of the sending FMU,
- `toFmuName`: the instance name of the receiving FMU,
- `fromVariableName` or `toVariableName`: the name of the variable as it is declared in the `ScalarVariable` section of the FMU.

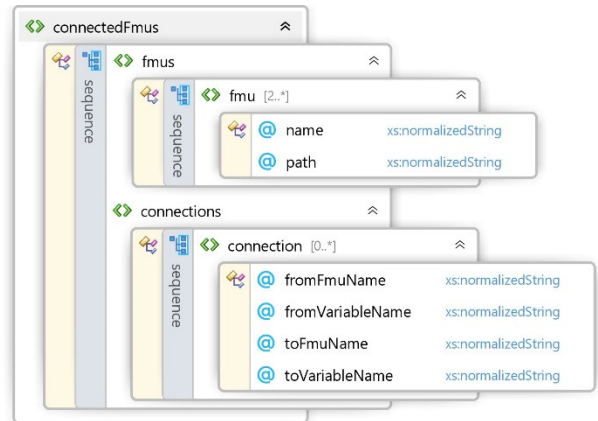


Figure 11. XML schema for connected FMUs.

If the units or the types of connected input and output variables are different, then this is automatically detected by the simulator before starting the simulation. For example, if `fromVariableName` is a Boolean variable and `toVariableName` is a Real variable, then the connection is not allowed and will be reported as an error.

5.2 Simulation Procedure

The new simulator plugin uses the existing FMU Simulator in PySimulator as a base. The simulator creates instances of the FMU Simulator classes depending on the FMUs defined in the XML file. In other words the FMUs are the component instances of the model. When the user adds the FMU, the simulator assigns a unique instance name to it. Thus, it is possible to have several instances of the same FMU. The simulator resolves the connections, i.e., getting and setting the values, between the time steps. From the point of view of the FMU simulator plugin it is just another FMU, thus the interface to the simulator is the existing FMU Python interface. Inside this Python interface the functions of the different FMU instances are called in the order defined by the connections. To determine the connection order evaluation, Tarjan's algorithm (Tarjan, 1972) is used. Algebraic loops are currently not supported. If there are no connections between the FMUs, then the order does not matter and each FMU is simulated independently.

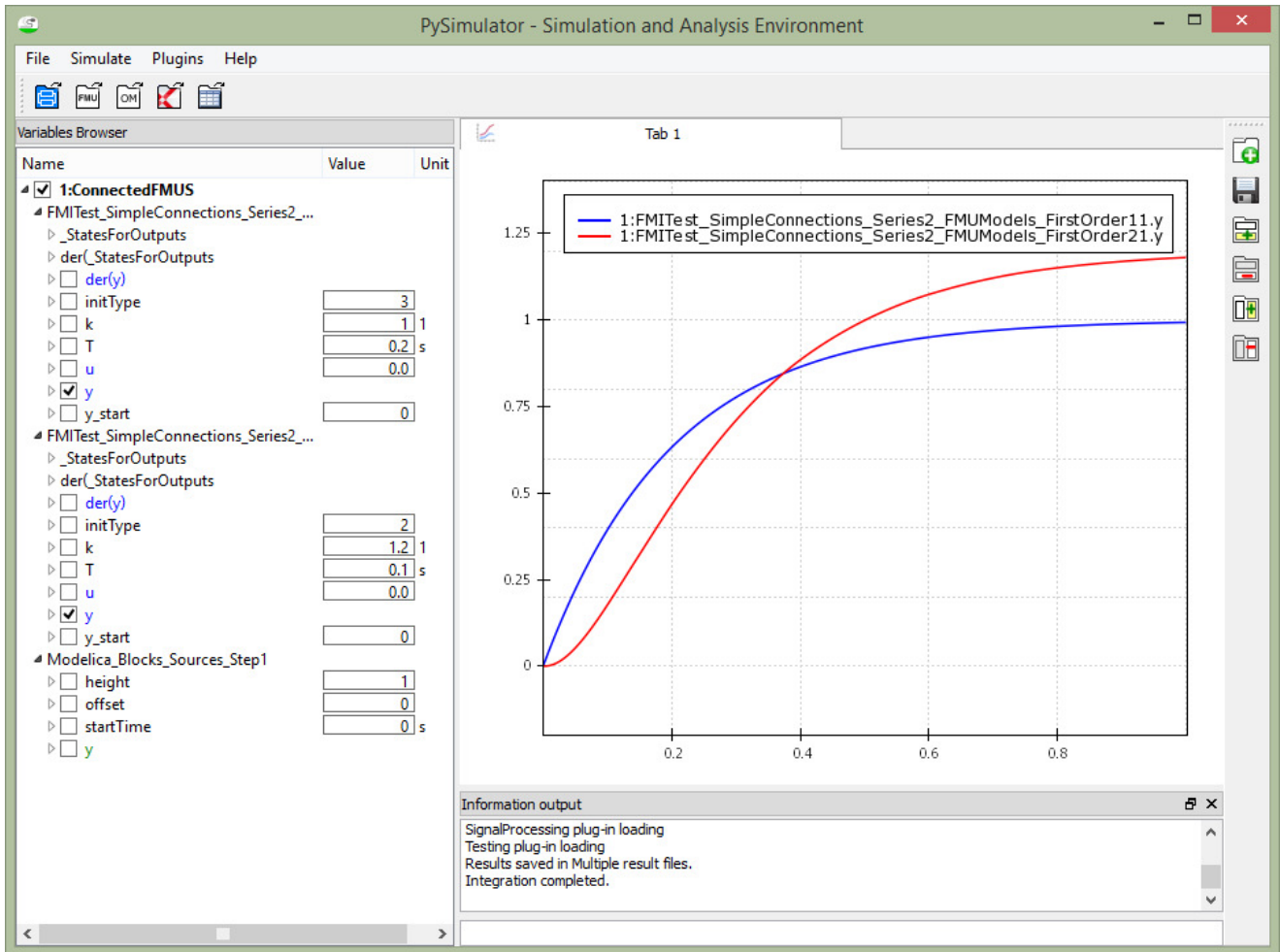


Figure 12. Simulation results of connected FMUs for Co-Simulation using an example from the Modelica Library FMITests.SimpleConnections.

The first prototype supporting the simulation of connected FMUs for Co-Simulation is complete. Some tests were performed using the Modelica library `FMITests.SimpleConnections` (Modelica Association, 2014b), see a plot of the results in Figure 12. The FMUs are generated using Dymola. The tests are also provided as part of PySimulator’s examples. The work on simulation of connected FMUs for Model Exchange is still under development.

6 Conclusions and Future Work

Comparing results of model simulation is very important for model portability and model evolution.

This paper presents a framework for regression analysis that can simulate models very efficiently and report how their results differ. Support for simulation of models in FMU form or using several Modelica tools including Dymola, SimulationX, and OpenModelica was previously present in PySimulator and has been extended in this work with a new simulator plugin for Wolfram SystemModeler.

Efficient regression analysis is provided by parallelization of model simulations and result comparisons.

A first prototype to simulate connected FMUs for Co-Simulation is complete. Ongoing work is focused on having fully functional simulation of connected FMUs for both Model Exchange and Co-Simulation.

The Modelica Association project *System Structure and Parameterization of Components of Virtual System Design (SSP)* aims at solving the problem where there is need to design, simulate, and execute a network of components. The project is in an early phase now but we might consider using its results to describe the connection of FMUs.

Acknowledgements

Part of the work is financed by the CleanSky Joint Undertaking project *PyModSimA* (JTI-CS-2013-2-SGO-02-064). This support is highly appreciated. Financial support of DLR by BMBF (BMBF funding code: 01IS12022A) for the FMU simulator in PySimulator according to FMI 2.0 within the ITEA2 project *MODRIO* (ITEA 2 – 11004) is also highly appreciated. The authors thank Jakub Tobolar (DLR Institute of System Dynamics and Control) for his tests and support of the regression testing feature in an earlier stage and his implementation of the automatic

generation of the simulation setup file by Dymola. The authors also thank Martin Otter (DLR) for the fruitful discussions about the topics presented in the paper.

References

- Benjamin Edwards. Pythonica, 2012. <https://github.com/bjedwards/pythonica> (accessed: 19th of May 2015).
- Anand K. Ganeson, Peter Fritzson, Olena Rogovchenko, Adeel Asghar, Martin Sjölund, and Andreas Pfeiffer. An OpenModelica Python Interface and its use in PySimulator. *Proceedings of the 9th International Modelica Conference*, 3.-5. Sep. 2012, Munich, Germany.
- ITI GmbH. Csv-compare tool, 2013. <https://github.com/modelica-tools/csv-compare> (accessed: 19th of May 2015).
- Modelica Association. Functional Mock-up Interface for Model Exchange and Co-Simulation, Version 2.0, July 25, 2014. <http://www.fmi-standard.org> (accessed: 19th of May 2015).
- Modelica Association. Functional Mock-up Interface. Subversion repository, 2014. https://svn.fmi-standard.org/fmi/branches/public/Test_FMUs/_FMIModelicaTest/FMITest (accessed: 21st of July 2015).
- Modelica Association. Modelica Standard Library 3.2.1, 2013. <https://github.com/modelica/Modelica/releases/tag/v3.2.1+build.2> (accessed: 30th of July 2015).
- Martin Otter. Private communication, 2015.
- Andreas Pfeiffer, Ingrid Bausch-Gall, and Martin Otter. Proposal for a Standard Time Series File Format in HDF5. *Proceedings of the 9th International Modelica Conference*, 3.-5. Sep. 2012, Munich, Germany.
- Andreas Pfeiffer, Matthias Hellerer, Stefan Hartweg, Martin Otter, and Matthias Reiner. PySimulator – A Simulation and Analysis Environment in Python with Plugin Infrastructure. *Proceedings of the 9th International Modelica Conference*, 3.-5. Sep. 2012, Munich, Germany.
- Andreas Pfeiffer, Matthias Hellerer, Stefan Hartweg, Martin Otter, Matthias Reiner, and Jakub Tobolar. System Analysis and Applications with PySimulator. *Presentation at the 7th MODPROD Workshop on Model-Based Product Development*, 4.-6. Feb. 2013, Linköping, Sweden.
- Python: multiprocessing — Process-based “threading” interface. <https://docs.python.org/2/library/multiprocessing.html> (accessed: 20th of May 2015).
- Robert Tarjan: Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, Vol.1, No.2, 1972.
- Wolfram: Wolfram Mathematica. <http://www.wolfram.com/mathematica> (accessed: 19th of May 2015).
- Wolfram: Wolfram SystemModeler. <https://www.wolfram.com/system-modeler> (accessed: 19th of May 2015).

Abrasive waterjet intensifier model for machine diagnostics

Gianni Ferretti¹ Michele Monno^{2 3} Bruno Scaglioni^{1 3} Massimo Goletti^{2 3} Marco Grasso^{2 3}

¹Dipartimento di Elettronica, Informazione e Bioingegneria DEIB, Politecnico Di Milano,
Via Ponzio 34/5, 20133 Milano, Italy

²Dipartimento Meccanica, Politecnico di Milano, via La Masa 1, 20156, Milano, Italy

³MUSP Lab, Via Tirotti 9, Le Mose, 29122 Piacenza, Italy

Abstract

This paper investigates the dynamics of a waterjet plant with multiple phased single-acting plungers. An object oriented dynamic model is proposed and discussed. The simulator may be tuned to generate signals under different health conditions to train multi-fault diagnosis tools. In fact, due to the challenging pressure conditions and the aggressiveness of abrasive materials, the reliability of machine tool components is a major concern. The information throughput provided by the model is validated with respect to real-industrial data, acquired in reference cutting scenarios.

Keywords: Waterjet Cutting; High Pressure Pump; Object-Oriented Modeling; Condition Monitoring

1 Introduction

Waterjet/abrasive waterjet (AWJ) cutting machines are used for several industrial applications thanks to the great flexibility of the technology, which is suitable for cutting a wide range of materials [Kovacevic et al., 1997]. This kind of machine tool includes an Ultra High Pressure (UHP) pump to generate the necessary pressure energy that is then converted into kinetic energy by the orifice into the cutting head. Different components, either belonging to the UHP pump or to the cutting head, are subject to different kinds of faults and performance degradation, due to the challenging pressure conditions and the aggressiveness of abrasive particles.

The reliability of AWJ cutting machines is therefore a topic of major concern in industry. A fast detection of a faulty state and the automatic identification of the root cause for observed symptoms are expected to provide several benefits, including the reduction of unexpected machine stops, a quick leakage recovery, the minimization of wastes, the enhancement of maintenance operations, etc. There are several studies in the literature

devoted to AWJ process monitoring [Peržel et al., 2012, Krenický and Rimár, 2012, Axinte and Kong, 2009, Rabani et al., 2012, Choi and Choi, 1997], mainly related to the determination and possible improvement of the cut quality. Nevertheless, very few authors investigated the development of automated tools for in-process monitoring and diagnosis of machine tool health conditions [Annoni et al., 2009, Grasso et al., 2013, Grasso et al., 2014]. One of major challenges consists of characterizing the AWJ plant behavior under both healthy and faulty conditions, in order to train fault classifiers.

Real data under faulty states are always difficult and expensive to collect, which makes purely data-driven diagnostic methods poorly attractive for a practical use. Model-based methods are expected to yield more effective diagnostics capabilities, thanks to the possibility of simulating the plant behavior under different operating conditions.

This paper investigates the dynamics of an AWJ plant with multiple phased single-acting plungers and it represents a first attempt to design an object-oriented dynamic model for such a kind of system. The model may be tuned to generate simulated signal patterns under different health conditions in order to train multi-fault diagnosis tool. Moreover, the model's behavior can be compared with the measurements and give indications on the failures, a diagnostic method based on object-oriented models is proposed in [Bunus and Lunde, 2008]. The proposed model generates simulated water pressure and plunger displacement patterns, which can be used to characterize the AWJ working cycle. The injections of degraded states and faulty conditions into the model allows to characterize the pattern deviations from the natural state, and hence to develop novel model-based fault detection and classification toolkits. The real industrial data include signals under healthy states and in the presence of faults affecting either the UHP pump components (cracked cylin-

ders) or the cutting head components (broken orifices). The paper is organized as follows: Section 2 describes the model, in Section 3 a comparison between experimental results and simulations is presented in case of normal and faulty conditions, and the model tuning is presented. Section 4 concludes the paper.

2 UHP intensifier model

In this section, the model of the CMS Tecnocut 60 HP intensifying machine is described, a picture of the device is reported in Fig. 1. This intensifier is able to reach more than 4000 bars by means of 3 single-acting cylinders with a maximum water flow rate of 5 l/min (orifices up to 0.4 mm in diameter). The machine scheme is reported in Fig. 2.

The intensifier is based on the Pascal principle: an oil circuit is pressurized by means of an electric pump, the oil flows into a single acting cylinder with a ratio of areas $1/v$ producing a pressure in the water circuit given by the input oil pressure multiplied by v . It must be pointed out that at least three cylinders are required in order to keep a quasi-constant output pressure, since the work cycle of the machine is composed by three strokes: the pre-compression stroke, where the outlet valve of the cylinder is closed and the oil flowing into the chamber heightens the water pressure, the compression stroke, where the outlet valve is open and the water flows from the cylinder to the orifice, and finally the back stroke, where the piston returns to the original position. The oil circuit is composed of two subcircuits, the high-pressure oil circuit depicted in black, and the back-stroke circuit depicted in red, the oil is pressurized by a variable displacement piston pump equipped with an hydraulic feedback control system.

It must be pointed out that the timing mechanism of the plungers is controlled by a PLC based on contactless proximity sensors. This mechanism, along with the displacement control system of the electric pump, plays a crucial role in the water pressure signal behavior. As will be seen later, the modeling phase of these components has been carried out in great detail.

The water circuit (blue in Fig. 2), is composed by a series of high pressure pipes connecting the outlet ports of the cylinders to the final orifice. The Modelica model of the intensifier has been carried out by means of the Hydraulics library [Hyd, 2014], which provided all the basic components for the model construction. In particular, fluid components for compressible water and oil are unavoidable in the application of interest, where the extremely high pressures compresses water by a factor

of 20%.

The fluids properties are made available to the hydraulic components by means of the `inner/outer` statements, hence the global machine model has been split in two main components, namely the water and oil circuits, where two different fluids have been adopted. The submodels are connected by means of translational 1D flanges, which represent the cylinders interfaces.

The water circuit, shown in Fig. 3, is composed by three chambers representing the part of the cylinders in contact with water, the supply circuit and the final orifice, modeled as an hydraulic resistance. It must be pointed out that the length and compliance of the pipe connecting the pressure intensifier and the final orifice are not negligible, hence a `longline` component has been used. The purpose of the aforementioned component is to model the dynamics of long pipes, such as the water hammer effect, and to consider the compliance of the pipe's walls subject to the water pressure. The final orifice was modeled by means of the `Orifice` model, a component of the Hydraulics library, where either laminar or turbulent flow can be used depending on the pressure drop on the component. The thermal effects on the orifice were not considered as the machine is self-coolant and no significant temperature gradient was detected, nor reported in literature.

The model of the oil circuit, shown in Fig. 4 is more involved, as it contains the oil pistons, the PLC, the group of mechanisms that feeds the oil to the cylinders and the variable displacement pump with its hydraulic feedback control system.

The oil feeding mechanism of the pistons is shown in Fig. 5. The circuit's valves are directly controlled by the PLC but the commutation delays are not symmetric with respect to the command signal, hence the delay is acquired as an external parameter, depending on the direction of commutation. It must be pointed out that valves commutation timing has great consequences on the pressure signal, therefore, accurate tuning of the delays is required. The influence of these parameters will be discussed in Section 3.

The model of the main oil circuit, including the pump and the displacement control system is shown in Fig. 6. The operating nominal pressure of the circuit, which directly controls the water pressure, is regulated by means of a relief valve that opens if the pressure exceeds the nominal value. The pump is equipped with an internal volume regulator, with the aim of optimizing the overall efficiency of the machine with respect to the various operation points. The regulator controls the volume of the pump by applying a pressure on the swash plate. The

volume is determined by the outflow rate through a system of valves and springs whereof parameters are not known. The dynamics of the control system was modeled as follows:

$$C_t(t) = \begin{cases} 1 & \text{if } Q_{out} < Q_s \\ 1 - \frac{Q_{out}(t) - Q_s}{Q_m - Q_s} & \text{if } Q_s < Q_{out} < Q_m \\ 0 & \text{if } Q_{out} > Q_m \end{cases} \quad (1)$$

$$C(s) = \frac{1}{(1 + Ts)} e^{-\tau s} C_t(s) \quad (2)$$

with $C(s)$ and $C_t(s)$ being the Laplace transforms of the control output $C(t)$ and the signal $C_t(t)$, while Q_s is the flow rate condition of maximum displacement and maximum operative pressure, Q_m is the flow rate at which the displacement of the pump is zero and finally T and τ are the parameters describing the inertia of the system. The parameters of the control system, as well as the valves' delays, were collected from the datasheets [Par, 2011].

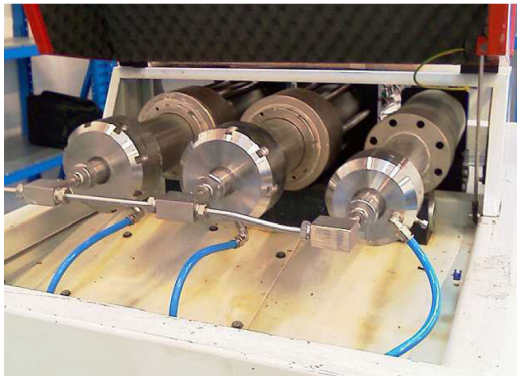


Figure 1: Pressure Intensifier

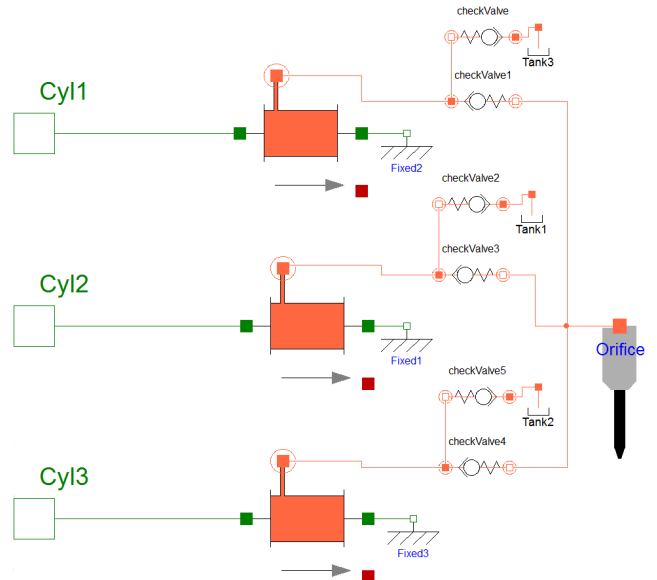


Figure 3: Water Circuit

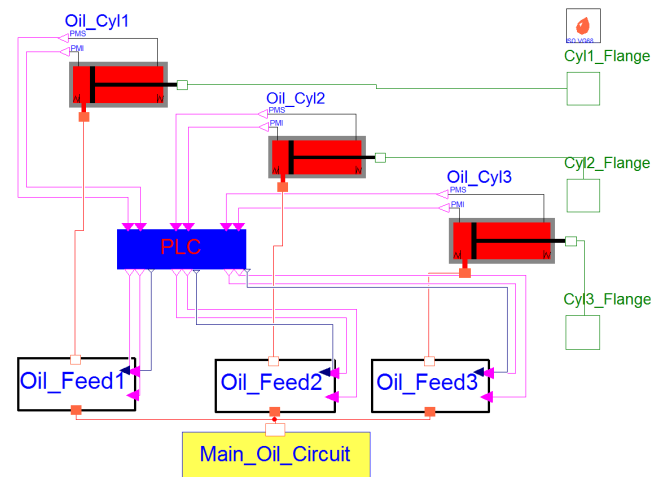


Figure 4: Oil model

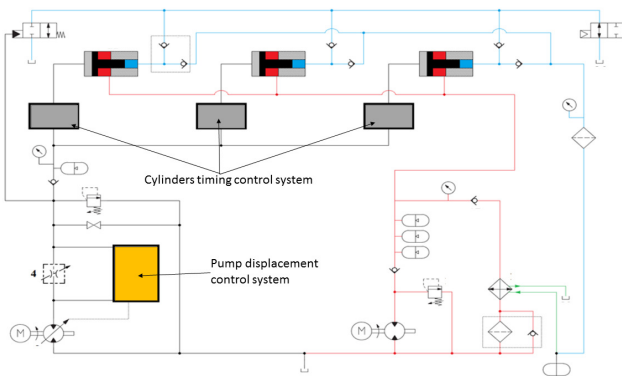


Figure 2: CMS Tecnocut UHP intensifier

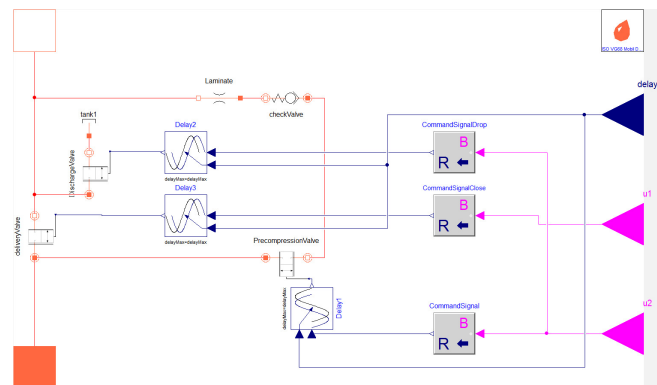


Figure 5: Feed circuit

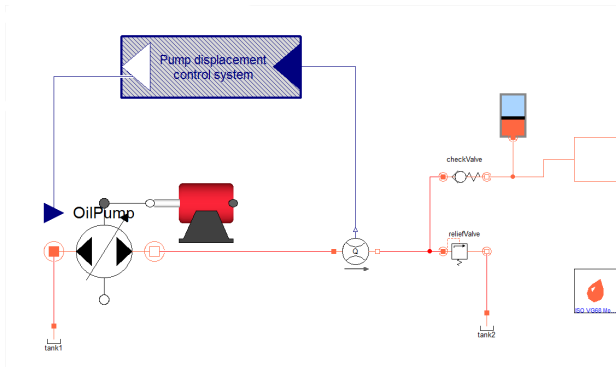


Figure 6: Main oil circuit

3 Simulation and model tuning

The model was simulated in the Dymola environment [Dynasim AB,] and validated with respect to experimental data. The experiments were carried out with a nominal water pressure of 3600 bar and a final orifice of 0.33 mm. The stroke of the three cylinders and the water pressure signal were acquired by means of a pressure sensor and three linear position transducers. It must be pointed out that the simulation of an hydraulic system characterized by pressures in the order of 10^7 Pa and flow rates of 10^{-6} m³/s can run into serious numerical stiffness problems. Moreover the commutation of the hydraulic valves can lead to situations where the pressure drop is close to zero in components with turbulent flow models, leading to numerical problems as the gain dQ/dP of the components goes to infinity as the pressure drop goes to zero, as described in [Hyd, 2014], hence the *Esdirk45a* integration algorithm has been used with a relative tolerance of 10^{-3} , which is a higher-order, A-stable algorithm suitable for stiff problems. It must also be pointed out that the classical *DASSL* algorithm often loses stability with tolerances between 10^{-3} and 10^{-5} , and could even not converge in reasonable time with smaller tolerances.

Fig. 7 shows the comparison between the experimental signal, depicted in red, and the simulated pressure signal, in blue. As it is apparent, the simulation is in good accordance with the experiments, the average pressure is similar as well as the in-cycle fluctuations, nonetheless experimental signals exhibit a decreasing trend with the periodicity of one cycle.

The observation is confirmed by the analysis of the power spectrum of the signals, visible in Fig. 8. A 1X-cycle component is visible in the experimental signal but it is missing in the simulated signal. The discrepancy was attributed to a difference between the cylinders' friction coefficients, whose parameter's values are

Index	Value
Simulated signal: Standard deviation	62.56 bar
Experimental signal: Standard deviation	42.86 bar
Correlation Coefficient	0.7147

Table 1: Model validation results

difficult to obtain. It must be pointed out that the Hydraulics lib implements a friction model described in [Tustin, 1947].

In order to fit the experimental data, the model was updated by assigning different values of viscous damping to the three cylinders. Figures 9 and 10 show the signal and the power spectrum with different damping values, the experimental pressure is faithfully reproduced and the power spectrum correctly identifies the 1X-cycle component. The standard deviation and correlation of experimental and simulated pressure signals were compared for the sake of model validation, and the results are shown in Table 3. Note in particular that the correlation coefficient between the signals is higher than 0.7.

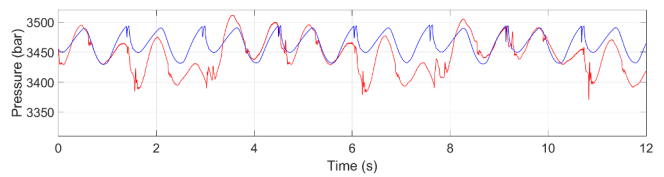


Figure 7: Real signal vs Simulated

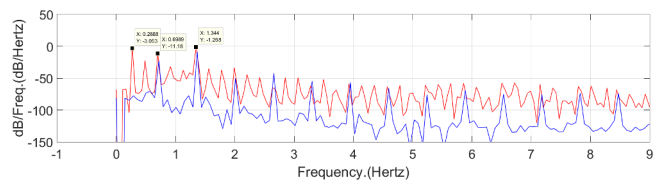


Figure 8: Power spectrum of experimental and simulated pressures

3.1 Faults injection

One of the main purposes of the developed model is to act as a virtual test bench for fault simulation, hence, a set of faults whose experimental data were available have been implemented in the model. Then, suitable indicators have been identified, based on the simulated and experimental signals.

One of the most frequent and important fault in the AWJ machines is a crack in the cylinder body, often

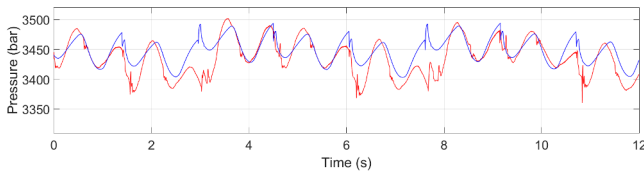


Figure 9: Experimental and simulated pressures with different friction coefficients

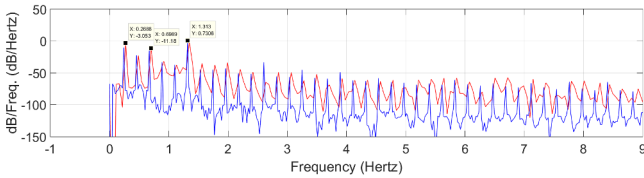


Figure 10: Power spectrum of experimental and simulated pressures with different friction coefficients

caused by the high pressures involved in the process. The fault was modeled as a leakage between the water chamber and the external environment and the conductance parameter was tuned on the advice of the cylinder manufacturer. It must be pointed out that the specific value of the leakage was tuned on the specimen under study, but the qualitative behavior of the system in fault condition is similar for a wide range of the conductance parameter. Fig.11 shows the cepstrum of the pressure signals. The Cepstrum of a signal [Childers et al., 1977] is defined as follows:

$$X(T) = |\mathbf{F}[\ln(|\mathbf{F}[x(t)]|)]| \quad (3)$$

Where $x(t)$ is the signal in the time domain, $X(T)$ is the cepstrum and \mathbf{F} is the Fourier transform operator. It represents the Fourier transform of the logarithm of the Fourier transform of the signal, and it is a good indicator of the main harmonics of the signal. In particular it is useful in the analysis of the water pressure as it clearly shows the length of the pumping cycle. The independent variable of a cepstrum analysis is the quefrency, measured in seconds. Note that the period of the pumping cycle decreases from 4.8 to 4.2 seconds in both cases, showing a good indicator for this kind of fault. Moreover, the strokes of the cylinders, shown in Fig. 12 exhibit a remarkable difference in the precompression stroke of the cracked cylinder with respect to the remaining ones.

The fault condition caused by a consumed final orifice was also investigated. The physical fault consists in a wear of the orifice, whose diameter increases and assumes irregular shape, with negative effect on the cutting quality. The fault was reproduced experimentally by installing a fatigued orifice on the machine, while

in the simulation environment, the diameter of the orifice was set to 0.35 mm according to the results of a geometrical analysis of the component. In the case of faulty orifice, a global reduction of the pumping cycle duration can be expected rather than a difference between the cylinders stroke signals. Fig. 13 shows the cepstrum of the simulated and experimental signal, the $1X$ – cycle component of the cepstrum reduces from a quefrency of 4.8 s to 3.2 s in the simulation and 3.3 s in the experiments, showing good accordance.

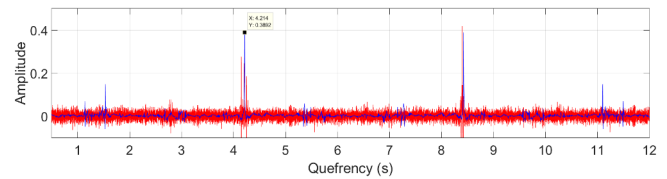


Figure 11: Cepstrum of the pressure signal, cracked cylinder

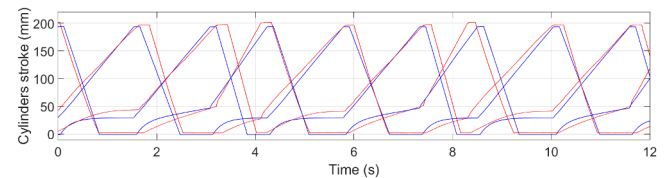


Figure 12: Cylinder strokes in case of fault

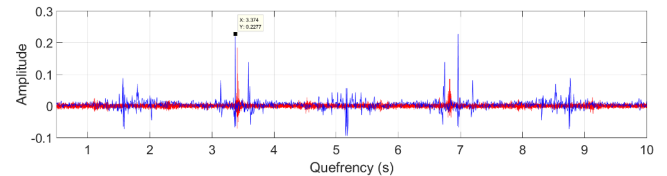


Figure 13: Cepstrum of the pressure signal, broken orifice

4 Conclusion

The object-oriented model of a complex machine tool, involving ultra high pressure water circuit, interaction between fluid mechanics and control system was presented. The model was tuned, and subsequently validated by means of a comparison with experiments exerted on a real industrial machine tool. Two different types of fault were introduced with the aim of reproducing the behavior of the machine in the case of malfunction. The developed model lays the groundwork for a model-based condition monitoring system which is expected to yield more effective diagnostics capabilities

by means of the on-line identification of faulty conditions. The development of the monitoring system will be addressed in a future work.

References

- [Par, 2011] (2011). *Parker Hannifin Corporation, Series DIVW, D Style catalog, USA: Parker.*
- [Hyd, 2014] (2014). *Modelon AB - Hydraulics Library - Version 4.1.*
- [Annoni et al., 2009] Annoni, M., Cristaldi, L., Lazzaroni, M., and Ferrari, S. (2009). Nozzles classification in a high-pressure water jet system. *Instrumentation and Measurement, IEEE Transactions on*, 58(10):3739–3745.
- [Axinte and Kong, 2009] Axinte, D. and Kong, M. (2009). An integrated monitoring method to supervise waterjet machining. *CIRP Annals-Manufacturing Technology*, 58(1):303–306.
- [Bunus and Lunde, 2008] Bunus, P. and Lunde, K. (2008). Supporting model-based diagnostics with equation-based object oriented languages. pages 121–130.
- [Childers et al., 1977] Childers, D. G., Skinner, D. P., and Kemerait, R. C. (1977). Cepstrum: A guide to processing. *Proceedings of the IEEE*, 65(10):1428–1443.
- [Choi and Choi, 1997] Choi, G. S. and Choi, G. H. (1997). Process analysis and monitoring in abrasive water jet machining of alumina ceramics. *International Journal of Machine Tools and Manufacture*, 37(3):295–307.
- [Dynasim AB,] Dynasim AB. *Dymola*. Lund, Sweden.
- [Grasso et al., 2013] Grasso, M., Goletti, M., Annoni, M., and Colosimo, B. M. (2013). A new approach for online health assessment of abrasive waterjet cutting systems. *International Journal of Abrasive Technology*, 6(2):158–181.
- [Grasso et al., 2014] Grasso, M., Pennacchi, P., and Colosimo, B. (2014). Empirical mode decomposition of pressure signal for health condition monitoring in waterjet cutting. *The International Journal of Advanced Manufacturing Technology*, 72(1-4):347–364.
- [Kovacevic et al., 1997] Kovacevic, R., Hashish, M., Mohan, R., Ramulu, M., Kim, T., and Geskin, E. (1997). State of the art of research and development in abrasive waterjet machining. *Journal of manufacturing science and engineering*, 119(4B):776–785.
- [Krenický and Rimár, 2012] Krenický, T. and Rimár, M. (2012). Monitoring of vibrations in the technology of awj. In *Key Engineering Materials*, volume 496, pages 229–234. Trans Tech Publ.
- [Peržel et al., 2012] Peržel, V., Hreha, P., Hloch, S., Tozan, H., and Valíček, J. (2012). Vibration emission as a potential source of information for abrasive waterjet quality process control. *The International Journal of Advanced Manufacturing Technology*, 61(1-4):285–294.
- [Rabani et al., 2012] Rabani, A., Marinescu, I., and Axinte, D. (2012). Acoustic emission energy transfer rate: a method for monitoring abrasive waterjet milling. *International Journal of Machine Tools and Manufacture*, 61:80–89.
- [Tustin, 1947] Tustin, A. (1947). The effects of backlash and of speed-dependent friction on the stability of closed-cycle control systems. *Electrical Engineers - Part IIA: Automatic Regulators and Servo Mechanisms, Journal of the Institution of*, 94(1):143–151.

Optimica Testing Toolkit: a Tool-Agnostic Testing Framework for Modelica Models

Anders Tilly¹ Victor Johnsson¹ Jon Sten² Alexander Perlman² Johan Åkesson²

¹Lund University, Sweden, {ada09ati, ada10vjo}@student.lu.se

²Modelon AB, Sweden, {jon.sten, alexander.perlman, johan.akesson}@modelon.com

Abstract

The need for regression testing increases as the size and complexity of software projects grow. The same is true for Modelica libraries and Modelica tools. Large Modelica projects often involves several Modelica tools and libraries which are under development. In those situations, with several orthogonal code bases, the need for systematic regression testing is needed.

In this paper we investigate a new way to create and run tests by developing a tool-agnostic testing framework. Additionally a graphical user interface for test authoring and management was created.

Keywords: Cross Testing, Testing Framework, Test Authoring, Regression Testing, User Interface, Modelica, FMI

1 Introduction

Optimica Testing Toolkit (OTT) is a tool-independent framework for performing automatic testing on Modelica models. It supports both static and script-based testing. Static testing is used to perform predefined tests on a subset of models in a library, where the user provides the specific library as well as a criteria for selecting which models to test. Each model is automatically compiled and simulated and the resulting trajectories are compared to reference trajectories. Script-based testing enables the test author to write finely tuned tests that interact with the compilation and simulation process and to test individual models with specific compiler and simulator scenarios. OTT supports cross-tool testing with several different Modelica compilers and simulation environments using FMI.

The purpose of OTT is not only to provide a framework for testing Modelica models, but also to provide a testing pipeline that is tool agnostic. OTT provides the same testing pipeline regardless of what compiler and simulator performs the actual model compilation and simulation. Tool agnosticism is provided by means of an abstraction layer between OTT and the actual tools. Each tool is hooked into the abstraction layer via a plugin tai-

lored specifically to that tool.

As part of the development cycle a Graphical User Interface (GUI) was developed (Tilly and Johnsson, 2015). The GUI can be used for test authoring, test configuration and test execution. One important aspect considered during development was to ensure that the GUI had good usability. We used a number of different user studies together with the users in order to discover usability problems, and then used iterative development to address and fix those issues.

OTT was initially developed by Modelon as an in-house tool for performing library testing and verification using several Modelica tools. It has since been extended with the GUI and other features and is now provided and maintained as a commercial product by Modelon¹.

2 Background

In software development, a test is usually run and checked towards an expected result (Burnstein, 2004). Testing Modelica models are tested using the same concept. More specifically, testing a Modelica model means testing if it: (a) can be translated and simulated without error, (b) delivers the expected results, and (c) represents reality adequately (Samlaus et al., 2014). For aspect b, there are reference values that are considered to be the “correct” values. The result of a test is checked to be within a specific tolerance of that value. If the modeler deems the new value to be better than the reference value, the modeler may choose to overwrite the old reference value and use the new value as future reference.

Testing consists of test authoring, test configuration and test execution. Authoring a test for a model means to select some variables to compare against references, and also changing some parameters in the model. Test configuration refers to choosing the appropriate settings for the test, such as which compiler to use, and test execution refers to running the tests.

In Modelica, models can be created and represented both textually and graphically. Using a graphical user interface is sometimes more efficient than using a program-

¹<http://www.modelon.com/>

```

model SimpleDeclaration
  extends Icons.TestCase;
  Real x = 3;
  Real y = x;
  annotation (
    __ModelicaAssociation(TestCase(
      shouldPass=true)),
    experiment(StopTime=0.01),
    Documentation(
      info="<html>Tests simple component  
declarations.</html>");
  end SimpleDeclaration;

```

Listing 1. An example of TestCase and experiment annotations. This example is taken from the Modelica Compliance Library Guide (Open Source Modelica Consortium, 2013).

matic approach (Chen and Zhang, 2007). Test authoring on the other hand is usually done programmatically.

The Modelica language contains the concept of annotations for storing meta information about the model. Examples of such information are: graphics, documentation and versioning (Modelica Association, 2014). There are two types of annotations that are relevant for testing:

- **experiment:** The experiment annotation indicates that the model can be simulated and it also provides simulation settings, such as start or stop time (Modelica Association, 2014). See listing 1 to see an example of this annotation.
- **TestCase:** The TestCase annotation extends the experiment annotations and specifies additional information, such as whether the test should pass or fail (Open Source Modelica Consortium, 2013). See listing 1 to see an example of this annotation.

2.1 Testing Frameworks

The testing process in software development is either automated or manual. Constructing an automated test is often more expensive than performing a single manual test. However, once the automated test has been specified, running it is much more efficient than performing the test manually. Because of this, automated testing is well suited for regression testing. Regression testing means performing tests continuously throughout the development process. This is done to discover possible introduced errors when making changes in the software. Manual testing on the other hand is done by a human. Manual testing is often required for GUI applications where how things look and feel is of interest. Performing automated tests for this purpose can be difficult.

Automated tests can be built and run using testing frameworks. A testing framework provides a way for specifying and executing tests. Some examples of established testing frameworks are:

- JUnit, a testing framework for the Java programming language (Gamma and Beck, 1999).
- Nose, a testing framework for the Python programming language (Arbuckle, 2010).

2.2 Usability

When we talk about usability in this paper, we mean the usability of software. Usability can be viewed as including a wide range of quality factors, for example maintainability. However, this paper focuses on the aspects of daily operation as defined by Soren Lauesen (2005). Lauesen defines usability to consist of six usability factors:

- **Fit for use:** Does the software have the needed functionality?
- **Ease of learning:** Is it easy to learn?
- **Task efficiency:** Is it efficient for the frequent user?
- **Ease of remembering:** How easy is it to remember for the occasional user?
- **Subjective satisfaction:** Does the user feel satisfied when using the software?
- **Understandability:** Does the user understand what happens in the software?

2.3 Related Work

Testing and automatic testing is nothing new to Modelica and FMI. Two examples of such implementations are: UnitTesting (Tiller and Kittirungsi, 2006) and MoUnit (Samlaus et al., 2014).

UnitTesting is a Modelica based library targeted at unit testing of Modelica models. Tests are created by defining Modelica models which extends the UnitTesting library. One big aspect of the testing library is to provide a wide range of metrics for the tested models. Example of supported metrics are component-, condition- and static-coverage.

MoUnit is a framework for automatic Modelica model testing. Tests are written in a language defined by MoUnit. MoUnit is integrated into the Modelica IDE OneModelica which supports the user during test authoring and test execution. However MoUnit can also be used standalone when integrated into automatic build environments such as Jenkins and Hudson. MoUnit provides result reporting and comparison against reference results.

The solution presented in this paper differs from these implementations. It is tool-agnostic, plugin-based and supports enhanced cross-testing. This enables library developers to verify their library with different Modelica and FMI tools. Additionally it allows for cross-testing between different compilation and simulation tools.

3 Test Methodology and Requirements

3.1 Static

Static testing is primarily used to test a large set of models that share one or more properties, e.g. package container, base class, annotation etc. A static test session begins with a set of Modelica packages containing test models as input, as seen in figure 1. The packages are traversed and models with properties matching a given set of criteria are selected. When a selection is made the test cycle; translation, compilation, simulation and verification begins. After verification is complete, information collected during test execution is passed to a set of output modules responsible for rendering the results, this completes the test cycle. The test session terminates when no more models are found.

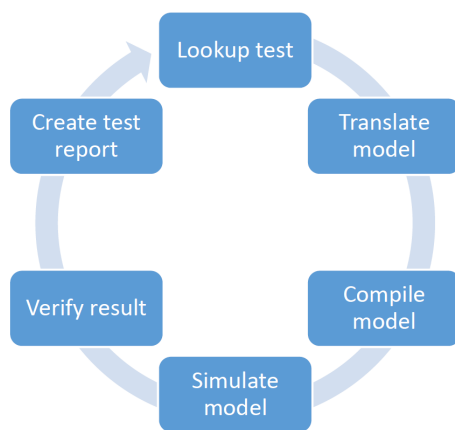


Figure 1. OTT static test cycle

The following types of static tests are currently supported:

- Experiments, tests models containing the `experiment()` annotation
- Test cases, tests models containing the `__ModelicaAssociation(TestCase())` annotation

A test session may be explicitly setup to run its test cycles up to and including a certain operation and still be considered completed (this is true of all static tests except for the Test cases tests where a test may be specifically designed to fail during one of the operations).

Due to the limitations of the experiment and test-case annotations, a new test specification format is under development. The format facilitates additional input and output for the test, such as modifiers for parameters, reference variables, tool specific options and much more. The current implementation of this test specification stores most of the information in an Extensible Markup Language (XML) file, but in some cases uses

other formats, such as tool-specific scripts and reference results. This paper will not explore this specification format further due to its current state.

3.2 Scripted

Script-based testing is primarily used to perform fine-grained and diversified testing of models which, unlike the models used for static testing, share none or very few properties. The OTT script-based testing pipeline gives the user total control over the testing process.

Much like static testing, OTT automatically retrieves relevant tests based on the sieve provided by the user. But that is where the similarities to static testing ends. It is the user defined test that is the driving factor during execution of scripted tests. Instead, OTT provide convenient and uniformed interfaces to the different Modelica tools and result reports. This gives the user full control of the test execution and less worry about tool specific interfaces. Additionally OTT provides mechanisms for populating and producing test reports.

Scripted tests are written in Python and resembles tests written for the Nose testing framework. However, unlike Nose, OTT provides interfaces to common Modelica and FMI tools.

3.3 GUI

The basic workflow when using the GUI is as follows: the modeller (a) creates a test for a specific model, (b) selects variables and parameters to include in the test, (c) runs the test and (d) examines the results.

When running a test in the GUI, the included variables and parameters and their values are extracted from the test and run using OTT. OTT then produces the results in the form specified, and if the results contain a HTML report, the report is displayed in the GUI.

The requirements for the GUI were that it should be user-friendly and it should provide all the necessary features. The workflow and features were discovered using user studies with the users, see 4.2.1 for more about this.

4 Implementation

OTT is a plugin-based tool written in Python and Java. It is able to interface to FMI and Modelica compliant tools either through Python interfaces or sub-process calls.

4.1 OTT Core

OTT Core contains functionality for collecting and performing static and scripted testing. It also contains abstraction layers for the different test steps, compilation, simulation and verification.

4.1.1 Overview

For static testing OTT uses the Optimica Compiler Toolkit (OCT)² to traverse Modelica libraries. During the traversal the user configurable sieve is used to collect tests by filtering the target library. Depending on the user's command OTT will then take the tests through the different test steps. Each test step provides one or more tool implementations. Current version of OTT supports OCT and Dymola both for compilation and simulation. Result verification is currently done using CSV Result Compare tool (CSV compare) developed by ITI GmbH, see figure 2.

Each test produces a test report containing information collected during execution. The test report is in an intermediate format which can be converted into any type of presentation format. OTT has a presentation layer which, like the tool abstraction layer, relies on plugins. OTT has the following set of default output plugins:

- HTML, produces reports in human readable form, see figure 3.
- JUnit, produces reports in machine readable form, suitable for build servers.
- Pickle (Python), produces serialized Python test report objects.
- Hash, produces a file mapping model names to hashed filenames.

The main entry point to OTT is through the MRTT command line program. It allows the user to specify a wide range of settings, such as: target library, what tools to use for the different steps, output type and tool specific settings. An overview of the OTT Core can be found in figure 4.

Scripted testing works in a similar fashion as static. However, unlike static tests, scripted tests control the execution flow. OTT only facilitates integration to supported Modelica and FMI tools. OTT also simplifies the generation of various report artifacts by providing access to the presentation layer. This allows the user to focus on authoring tests instead of writing report files, such as HTML and JUnit reports.

4.1.2 Jenkins Integration and JUnit

OTT can easily be integrated into common Continuous Integration (CI) frameworks such as Jenkins and Hudson. This is done by configuring OTT to output a JUnit test report. This report is then parsed by the CI framework. The JUnit report contains status information for the different test steps, each with its own pass/fail flag. This enables the framework to detect changes in tests that changes between two failing states, i.e. if a model goes from compilation failure to simulation failure.

²<http://www.modelon.com/>

4.2 GUI

The OTT GUI allows the user to create, modify and execute tests. It was developed with usability in mind to ensure that it would be user-friendly.

4.2.1 User Feedback

We continuously evaluated the GUI by using the methods described by Lauesen (2005). Every iteration began with an evaluation of the GUI in the form of a user study followed by a response in the form of implementation in the GUI. The features that were implemented often directly addressed some usability concern.

Here are some important usability concerns we addressed:

- How to find the names of variables and parameters that will be included in the test.
- How to update the reference value of a test.
- How to view the results of the test.
- How to create many similar tests, and how to update them.

4.2.2 Features

In the GUI, as seen in figure 5, variables and modifiers (parameters) are displayed in tree views. Every tree view has a filter to make it more flexible to navigate the view.

The GUI has support for test inheritance, primarily to make it easier to create many similar tests. Test inheritance means that a subtest can be created to an already existing test. The subtest inherits all of the included variables and modifiers of the parent. The subtest can then change the value of those modifiers and add additional modifiers or variables. If the parent tests is updated all subtests will be updated. For example, as seen in figure 5, test c is a subtest of test a. Test a includes the modifiers `driveAngle` and `inertial.J`. Test c inherits these two modifiers and also changes the value of `inertial.J`. Test c also includes its own modifier `inertia2.J`.

After a test or suite of tests are run, the results will be displayed in the GUI. Reference results can be updated by pressing a button in the displayed results file. This allows the modeler to overwrite the old reference value if the new value is deemed more appropriate. When updating the reference, all variables specified in the test are updated.

Some basic features included in the GUI are: undo/redo operations, keyboard shortcuts and a run configuration.

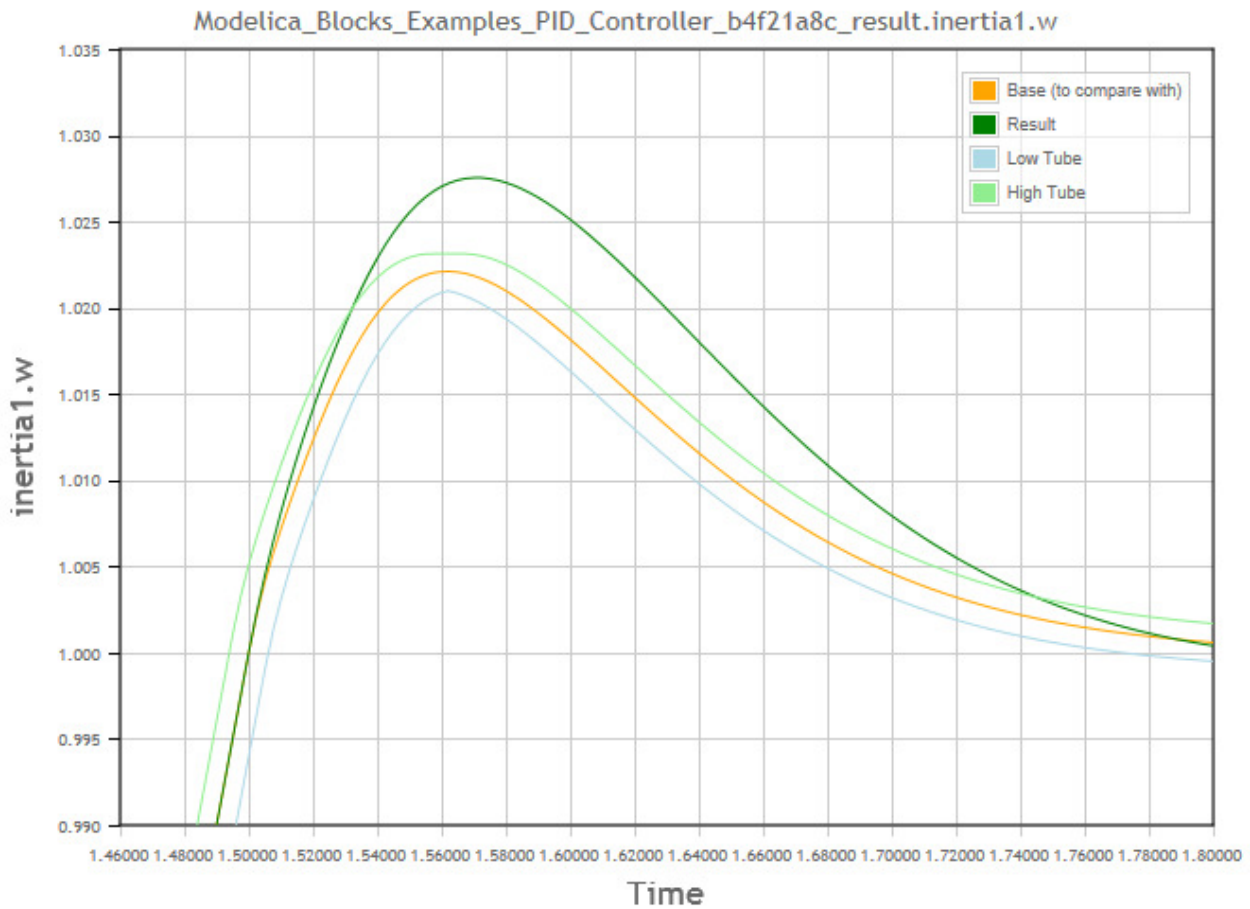


Figure 2. One variable compared to the reference value in the result file.

Verification Report

Modelica.Blocks							
Models	Compilation	[s]	Simulation	Time [s]	n	s]	Rate
Modelica.Blocks.Examples.PID_Controller	pass	4.46	pass	0.41	pass	1.91	100%
Modelica.Blocks.Examples.Filter	pass	5.81	pass	0.92	fail	n/a	Either the result file or the reference file does not exist
Modelica.Blocks.Examples.FilterWithDifferentiation	pass	5.53	pass	0.48	pass	0.96	100%
Modelica.Blocks.Examples.FilterWithRiseTime	pass	4.69	pass	0.36	pass	1.11	100%
Modelica.Blocks.Examples.InverseModel	pass	3.29	pass	0.46	pass	0.72	100%
Modelica.Blocks.Examples.ShowLogicalSources	pass	3.34	pass	0.27	pass	0.42	100%
Modelica.Blocks.Examples.LogicalNetwork1	pass	3.37	pass	0.35	pass	0.48	100%
Modelica.Blocks.Examples.RealNetwork1	pass	3.40	pass	0.41	pass	0.91	100%
Modelica.Blocks.Examples.IntegerNetwork1	pass	3.64	pass	0.41	pass	0.87	100%
Modelica.Blocks.Examples.BooleanNetwork1	pass	4.25	pass	0.40	pass	1.11	100%
Modelica.Blocks.Examples.Interaction1	pass	4.25	pass	0.41	pass	0.95	100%
Modelica.Blocks.Examples.BusUsage	pass	3.32	pass	0.35	pass	0.57	100%
Summary	Passed compilation: 12/12		Passed simulation: 12/12		Passed verification: 11/12		

Figure 3. The results from testing a package with OTT.

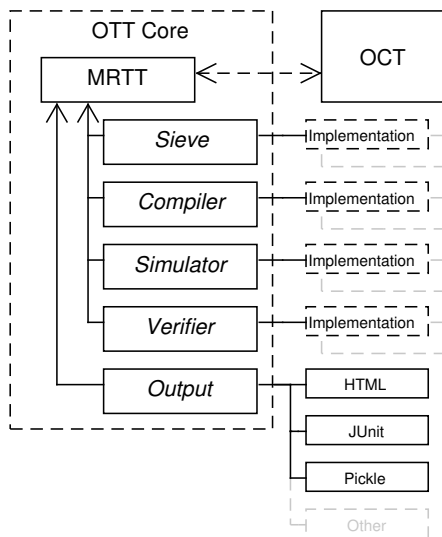


Figure 4. Overview of OTT Core

5 Conclusion and Future Work

In this paper, we presented a solution for tool agnostic regression testing in Modelica. By using a plugin like structure we have shown that it is possible to achieve a clear separation between the different testing steps. This allows us to use one Modelica tool to compile the model and another FMI tool to simulate the model, thus providing true cross-testing of Modelica libraries and tools. We have also shown why both scripted and static testing is necessary in Modelica development and how it can be implemented in a testing framework. The plugin structure also facilitates extendable and customizable test reports. One demonstration of this extensibility was exemplified by showing the integration to automatic build systems such as Jenkins/Hudson by creating an output module that writes JUnit reports.

Additionally we present a GUI which enables the user to do test authoring, test execution and viewing of test results. We show how the GUI can improve the efficiency of test authoring by providing tools for efficient selection of test variables and parameters in the test model. We also show how the usability of the GUI was improved using iterative user studies and development.

In the future we plan to extend the number of supported Modelica and FMI tools, which will further strengthen the cross-testing capabilities. In order to improve the usability of the GUI we plan to integrate a graphical model viewer that has previously been implemented (Sten, 2012). Likewise we plan to render model icons correctly by integrating a previously developed icon rendering framework (Olsson and Moraesus, 2011).

References

- Daniel Arbutckle. *Python Testing: Beginner's Guide*. Packt Publishing Ltd, 2010.
- Modelica Association. Modelica - a unified object-oriented language for systems modeling, language specification version 3.3 revision 1. page 31, 2014.
- Ilene Burnstein. *Practical Software Testing : A Process-Oriented Approach*. Springer, 2004.
- Jung-Wei Chen and Jiajie Zhang. Comparing text-based and graphic user interfaces for novice and expert users. In *AMIA Annual Symposium Proceedings*, volume 2007, pages 125–129. American Medical Informatics Association, 2007.
- Open Source Modelica Consortium. *Modelica Compliance Library Guide*. 2013.
- Erich Gamma and Kent Beck. Junit: A cook's tour. *Java Report*, 4(5):27–38, 1999.
- ITI GmbH. Csv result compare tool. <https://github.com/modelica-tools/csv-compare>. Accessed: 2015-05-19.
- Soren Lauesen. *User Interface Design - A Software Engineering Perspective*. Addison-Wesley, 2005.
- Kristina Olsson and Lennart Moraesus. Eclipse-based graphical rendering and editing of modelica code. Bachelor's Thesis, Lund University, 2011.
- Roland Samlaus, Mareike Strach, Claudio Hillmann, and Peter Fritzon. *MoUnit - A Framework for Automatic Modelica Model Testing*. Proceedings of the 10th International Modelica Conference, 2014. doi:10.3384/ecp14096549.
- Jon Sten. Graphical editing in jmodelica.org. Master's thesis, Lund University, 2012.
- Michael M Tiller and Burit Kittirungsri. *UnitTesting: A Library for Modelica Unit Testing*. 2006.
- Anders Tilly and Victor Johnsson. Developing a test authoring tool for a modeling language. Master's thesis, Lund University, 2015.

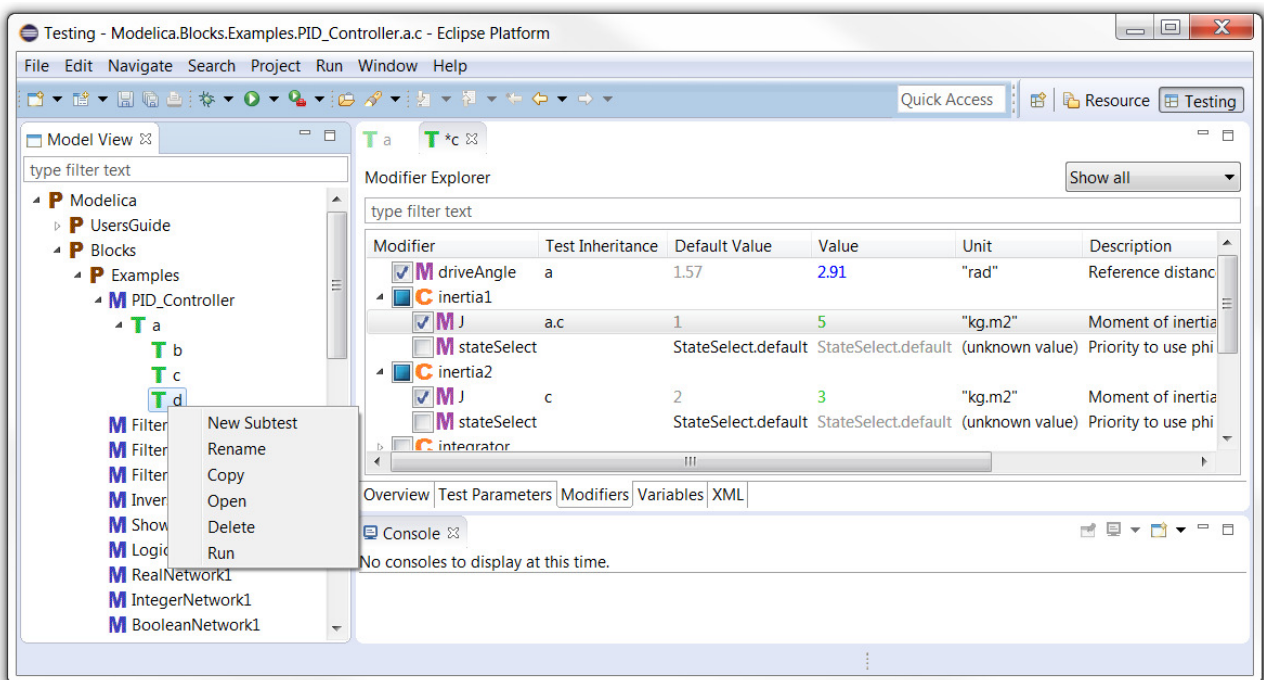


Figure 5. The OTT GUI

Status of the TransiEnt Library: Transient simulation of coupled energy networks with high share of renewable energy

Lisa Andresen¹ Pascal Dubucq² Ricardo Peniche Garcia³
Günter Ackermann² Alfons Kather³ Gerhard Schmitz¹

Hamburg University of Technology, Am Schwarzenberg-Campus 1, Hamburg, Germany

¹Institute of Thermo-Fluid Dynamics, {andresen, schmitz}@tuhh.de

²Institute for Electric Power Systems and Automation, {dubucq, ackermann}@tuhh.de

³Institute of Energy Systems, {peniche, kather}@tuhh.de

Abstract

The Modelica library `TransiEnt` is being developed within the research project *TransiEnt.EE*. After completion, the library will be freely available and will provide a framework to model coupled energy supply grids, i.e. electricity, district heating, and gas grids, including their corresponding producers, consumers and storage systems. This paper presents the current status of the library and outlines the library's structure and the modeling concept. The application possibilities of the library are presented in an exemplary simulation where the city of Hamburg is selected as the reference system. The impact of a high share of fluctuating renewable energy generation in the electric grid and the integration of excess electricity in the district heating network is presented.

Keywords: coupled energy grids, electricity, district heating, gas, dynamic simulation, renewable energy

1 Introduction

In 2011, the European Union set ambitious emission reduction targets to contribute to the abatement of climate change. These targets pursue a 80 to 95 % reduction of greenhouse gas emissions by 2050 compared to the 1990 emission levels (European Commission, 2011). To reach this goal, Germany aims to cover 80 % of its gross electric energy consumption with renewable energies (RE) by the year 2050 (BMWi, 2014). However, the timely offset between RE generation and electricity demand, together with limited inter-regional electricity transport capacities leads already to regional imbalances, which results in RE curtailment.

The so-called energy triangle that illustrates the main goals of energy policy considers not only environmental protection, but also economic efficiency and security of supply. With Germany's *Energiewende* the objective of less greenhouse gas emissions is heavily promoted. This must not lead to an unaffordable and unreliable energy

supply system. Thus, all three objectives should be taken into account when comparing different future scenarios. Consequently, simulations are necessary.

The research project *TransiEnt.EE* (Hamburg University of Technology, 2013-2016) currently being executed at the Hamburg University of Technology has two main objectives: a) to analyze and compare different strategies for the integration of renewable energies in urban energy systems considering transient effects derived from coupling of energy grids and b) to develop a freely available Modelica library which allows this kind of studies. The project started in May 2013 and will be finished in October 2016.

After completion, the library `TransiEnt` (Transient Energy Networks) will be made available under the terms of the Modelica license agreement. The current development is performed using Dymola (Dassault Systemes, 2012). Modelica was chosen as the developing language because it allows multi-domain simulation, which is handy when simulating coupled electricity, district heating and gas grids. Besides, its object-oriented features simplify the development process and usability of the library.

This paper is structured as follows: first, the technical background of coupled energy grids is presented. Afterwards, the library's package structure and modeling approach are presented, together with a brief description of other used Modelica libraries and the definition of the main interfaces. Finally, an example simulation is presented which shows the applicability of the library.

2 Technical Background

The `TransiEnt` library covers the whole energy infrastructure with its corresponding producers, consumers, grids and storage systems. A brief description of these and the properties and basic differences of the electricity, heating and gas networks will be given.

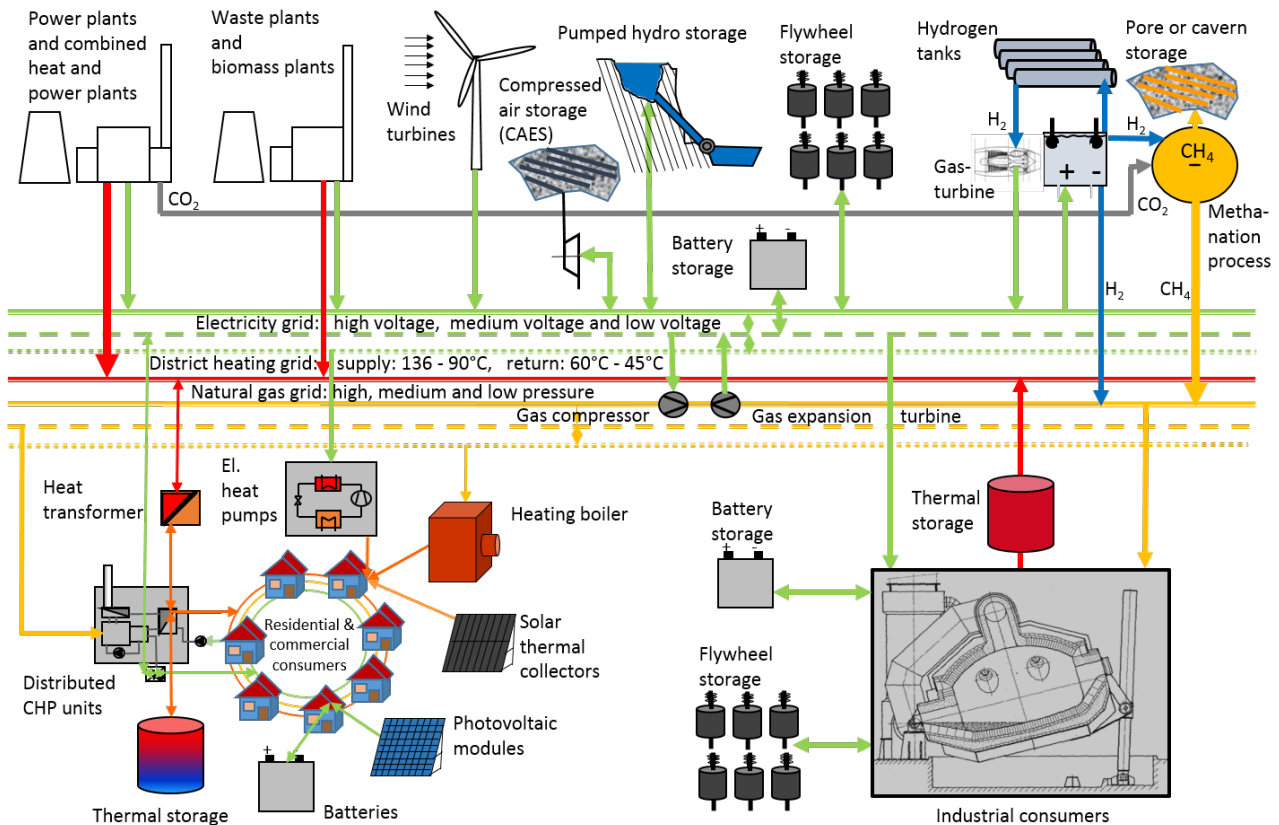


Figure 1. Scheme of coupled electricity, district heating and gas energy supply systems including their corresponding small- and large-scale producers, consumers, and storage systems.

2.1 The Energy Supply Infrastructure

The energy supply infrastructure can basically be grouped into four parts: the producers, the distributing grids, the consumers, and the storage systems, which are briefly defined in the following:

- Producers are small- to large-scale technical units that convert energy into electric power, and / or thermal power.
- Grids serve for transporting and distributing energy from the sources or producers to the consumers.
- Consumers devalue the final energy by "using" it. They are characterized by electric and thermal power loads.
- Storage systems are technical units that accumulate a form of mechanical or inner energy. They can be localized at any point of the energy supply infrastructure and serve for timely decoupling of production and utilization.

Figure 1 shows schematically the electric, heat and gas supply infrastructure with its participants. In the top third of the scheme, large scale fossil and renewable power plants, as well as central storage units are shown. In the

middle, the three distributing grids are displayed. Furthermore, different consumers, smaller storage units, and decentralized energy converters are shown in the bottom of the scheme.

As *coupled* energy grids are the purpose of modeling, the links between the grids shall be described briefly in the following. Some of them are well-known and defined by the conversion of gas to electric or thermal power or both. For consistency reasons these shall be called gas-to-power, gas-to-heat, and gas-to-power-and-heat technologies, respectively. Recently, other connection technologies are in development and tested in pilot plants. These are often called power-to-heat and power-to-gas technologies. The name indicates the direction of energy conversion. Power-to-gas units use electric power to split up water into hydrogen and oxygen. The hydrogen can be further deployed in the methanation process to obtain methane. Both products can be used directly or fed into the natural gas grids. Power-to-heat units convert electric into thermal power, either directly by using resistance heaters or electrode boilers or thermodynamically more efficient by the use of heat pumps. The thermal energy can then be used directly or fed into the district heating grid. Both, power-to-gas and power-to-heat systems, could be used in the future to adapt and increase the amount of electrical renewable generation used.

2.2 Differences and Dynamics of the Grids

There are some differences between the electricity, heating, and gas infrastructures. To begin with, natural gas is a primary energy carrier and needs to be converted to be usable, whereas in the heating and electricity grids the transported energy has already gone through at least one lossy conversion. Another difference is that in heating and gas grids energy can be stored to some extent. In the electricity grid, on the other hand, almost no capacity is available for storage. Thus, demand and supply have to be equal at any time, which makes the stability of the electricity grid a challenging task. This is why there exists an European integrated network coordinated by the ENTSO-E (ENTSO-E, 2015) to balance skew positions supra-regionally, in an efficient and fast way. Consequently, the electricity networks of cities or even countries are not isolated, which has to be considered in the modeling. This is different for district heating and gas grids. Heating grids only supply smaller regions at most as big as cities. The thermal power is produced by local (combined) heat (and power) plants corresponding to the local demand. Germany's gas economy highly relies on imports, which normally do not coincide with the demand but are done according to the price, storing gas during summer and releasing it in winter. Delivery bottlenecks are rather seldom and are not accounted for in the modeling. Gas markets are regional and the chemical composition of the gas mixture differs regionally and could be more and more influenced by local biogas and hydrogen feed-in in the future. Therefore, heating and gas grids need to be modeled with mass transfer and local sources and sinks to some extent.

As the name of the library suggests, *transient* simulations of the coupled energy networks are carried out. This is to account for ramp rates, reaction times, and dead times of the coupled system's components. The need for dynamic simulations shall be explained by two examples. For instance, the off-take of gas from the grid is not constant during the year or even the day. Consequently, the maximum volume flow rate of hydrogen - only a defined Vol.-% H₂ is allowed - fed into the natural gas grid varies. Thus, there could be moments with excess renewable power where the gas grid is not able to take up more hydrogen. Another example concerns the inertia of the heating grid. The propagation velocity of a change in temperature in the district heating grid is approximately 8.7 km/h for a heating water mass flow of 3300 t/h (Chudzienski, 1987). This means, that a sudden rise in thermal power consumption takes around 2.3 hours to be noticeable by means of a lower return temperature at the production plant in 20 km distance. In the electricity network, on the contrary, changes in demand are noticeable instantaneously. The model has to account for these different time constants.

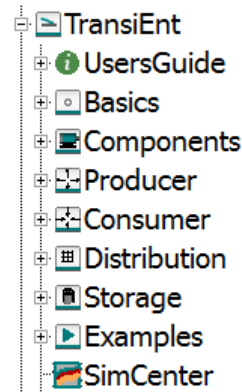


Figure 2. Top level tree of the `TransiEnt` library package.

3 The TransiEnt Library

The broad range of physics and fields considered in the `TransiEnt` library demands some effort in readable modeling and an user-friendly structure. Moreover, the level of detail (LoD) of the models has to be suitable for the task of simulation: complex single and coupled energy supply systems with various producers, consumers, storage systems, and distribution networks as well as their links for areas as big as cities.

In the following, the content of the library package is described, the modeling principles is explained, and a brief introduction to other used Modelica libraries as well as details to the main interfaces are given.

3.1 Library Structure

Figure 2 shows the top level tree of the `TransiEnt` library. The layout of the library was chosen carefully in the purpose of reuse-ability by applicants as well as expand-ability by model developers. The `Basics` package contains supporting classes like functions, units, blocks, media records, data tables, icons, and interfaces. Single components, e.g. electrical machines, pumps, and pipes, are structured within the `Components` package. These first two packages are meant to be used and modified by advanced users and developers only, whereas the remaining packages are meant for users that want to build up energy systems in order to examine different scenarios. These packages are named after the four participating groups in the energy supply system: producer, consumer, distribution, and storage. Within the `Producer` package there are small and large, conventional and renewable plants converting primary energy into electric work, heat or both. The `Consumer` package comprises models of electric power and thermal power loads for households, commercial buildings, industry as well as for bigger areas like city districts or whole cities. `Distribution` is composed of electric, heat, and gas distribution elements. Within the `Storage` package there are different power-to-power (e.g. pumped

hydro storage), heat-to-heat (e.g. sensible water storage tanks), gas-to-gas (e.g. caverns), as well as power-to-heat and power-to-gas converters. The last package `Examples` contains examples for the different system models in general and for the examined system of the city of Hamburg. These examples allow the users to easily understand the usage of components and the scope of application of the library.

3.2 Global Parameters and Statistics

A basic concept that was partly taken from the `Clara` library (Clara Library, 2015) and was extended to the requirements of the `TransiEnt` library is the concept of the global `SimCenter` model. This model has basically three purposes within the `TransiEnt` package:

- setting global parameters for the three energy supply infrastructures, e.g. the media in the gas and heat grids or the nominal frequency in the electric grid,
- choosing ambient condition time lines, which can be changed and extended easily by extending a modified `MSL CombiTimeTable` and setting the path to the data file in ASCII or MAT-file format, and
- collecting simulation statistics, i.e. the produced and consumed power and energy, the costs for heat and electricity, and the CO₂ emissions, each statistic variable calculated in every time step.

Each sub-component within the `TransiEnt` package has an outer instance of the model `SimCenter` called `simCenter`. The executable model then has to have an inner `simCenter`. Here, the settings for the simulated system and all its objects are defined and the statistics are collected.

To avoid the need of post-processing steps for the analysis of scenarios the `SimCenter` model provides the user with summarized results of the simulated energy system. For this purpose, it contains four gathering blocks: electric power, heating power, incurred costs and emissions. Each block gathers information from distributed sensors located within the component models and allocates them to specific types, e.g. total produced energy by RE.

3.3 Modeling Principles

In general, the principles of the modeling within the `TransiEnt` package can be summarized as follows:

- flat hierarchy for high readability,
- high flexibility by means of exchangeable models with different physical effects considered, and

- easily changeable boundary conditions.

The modeling principles mainly arose due to the complexity of a coupled energy supply system, containing systems with very different time constants (s. section 2) and also control units. Simulating a big coupled system is a challenge, which is why the systems can be built up of simple models (with low level of detail) in the first instance. If the analysis of the results shows that some effects should be more detailed, this can be done by replacing the simple model by a more advanced one. The choice of the physical effects and also dynamics to be considered highly depend on the question to be answered. On the other side, the level of detail (LoD) is restricted by the number of equations and computing power, respectively. For instance, the model resolution can be higher if only a city district shall be simulated and not a whole city. One of the questions to be answered within the scope of the `TransiEnt.EE` project is a about the level of detail of the subsystems. Since the accuracy of the whole system depends on the accuracy of the subsystems, the LoD of the latter have to be in such a way that all physical and transient effects that have a considerable impact on the overall results (CO₂ emissions) are modeled. On the other hand, the simulation of the whole system should still be computationally manageable within reasonable CPU times.

3.4 Level of Detail

As mentioned before, the purpose of the `TransiEnt` library is not the optimization of a single component of the system, e.g. the efficiency of a battery stack, but rather the optimization of the interaction of various components in order to improve the integration of renewable energies. Therefore, there exists a certain limitation in the level of detail of the subsystems that won't be exceeded.

Besides distinguishing between static and dynamic models, the modeling approaches can generally be classified into table-based models, models that are based on characteristic lines and / or transfer functions, spatially averaged, and spatially discretized balance equation-based models. Table-based models are based on the `CombiTimeTable` and require data input. Some electric and thermal power data for consumers and electric power data for renewable energy plants are provided in quarter hourly resolution for Hamburg.

For the main participating groups (s. section 2.1) the following approaches are implemented:

- Producer: table-based models, transfer function-based models, balance equation-based models,
- Grids: electrical grid only considers power flow and center of inertia grid frequency; heating and gas grids are built up of pipes that can be spatially discretized and use real fluid mixtures to consider evaporation in the heating network and hydrogen feed-in in the high pressure gas grid,

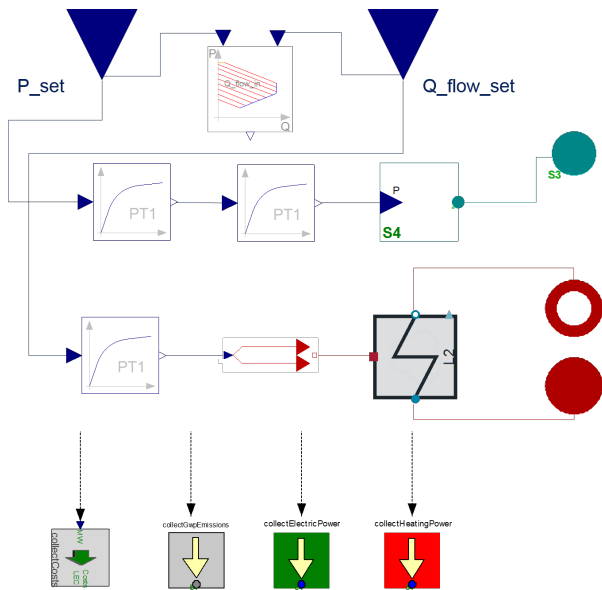


Figure 3. Example of a subsystem composed of models with different levels of detail.

- Consumer: table-based models, balance equation-based models,
- Storage: transfer function-based models, balance equation-based models.

The model complexity of coupled urban energy systems leads to a need of simplifying the subsystem models. A trade-off between simplicity and high LoD is required. In the *TransiEnt* library, this is achieved by combining table-based, transfer function-based and balance equation-based models. To illustrate this concept, a very simple combined heat and power (CHP) plant model is shown in Figure 3. The incoming set values for the electricity and heat production are received from external schedulers via real input connectors and then transferred to a table-based model and to first order blocks. The table-base model in this example is a look-up table with plant-specific heat input rates. These values can then be used to calculate load-dependent efficiency values. On the other hand, the first order blocks can be parameterized to represent plant-specific power gradients. A balance equation-based heat exchanger model from the *Clara* library is used in this example to calculate the heat transfer to the district heating water. Finally, the four collecting sensors at the bottom of the figure calculate generation costs, emissions and generation values and transfer these results to the inner *simCenter* instance.

More detailed combined models and models based solely on balance equations are currently under development. The later models can be used for detailed analysis of smaller time-frames. However, their complexity makes their usage for annual simulations of coupled energy systems impractical.

3.5 Usage of Other Libraries

Besides the Modelica Standard Library (MSL), *TransiEnt* mainly uses two other libraries, that will be introduced briefly.

The free Modelica library *Clara* (*Clara* Library, 2015; Brunnemann et al., 2012) was released in March 2015 and is the main product of the research project *DynCap* (Kather et al., 2014). *Clara* (Clausius Rankine) models the transient thermal behavior of power plants and power systems. The library contains all components of the water-steam cycle and the gas treatment path of power plants. Basic components like pipes and heat exchangers are used in the *TransiEnt* library in the heating systems. Power plant models with a low level of detail can be used as well. *Clara* does not use media models from the MSL but from the *TILMedia* Suite.

TILMedia (TLK Thermo GmbH, 2015) is an interface library to provide thermophysical properties from various existing fluid and solid property databases as well as own implementations to different applications, e.g. Modelica/Dymola or Matlab/Simulink. Media models used in *Clara* are provided and can be used freely. Further necessary models for the *TransiEnt* library (e.g. natural gas) will also be freely available. For other implementations beyond the scope of the *TransiEnt.EE* research project a supplementary license will be necessary.

TILMedia was chosen due to better numerical performance and robustness compared to *Modelica.Media* which was tested during the *DynCap* project (Brunnemann et al., 2012; Kather et al., 2014). These factors are regarded as critical for simulation of big coupled and closed loop systems considered here. Furthermore, *Modelica.Media* so far does not provide media properties for real gas and fluid mixtures.

3.6 Interfaces

There are two different types of interfaces: one for fluids and one for electric terminals. The fluid interface is taken from *Clara* - since most basic components (e.g. pipes for the heating grids) are used and extended from here. The medium model in the connector is an extension of the *TILMedia* class *BaseVLEFluid* (s. Listing 1). For real fluid behavior this type of medium class is favored. For ideal gas behavior (e.g. assumed for exhaust gas), a *GasTypes* class from *TILMedia* is used (not displayed here). An adapter to *Modelica.Fluid* is provided in *Clara*.

The second type of interface is for the electric systems. The main interface has two variables: active power and frequency (s. Listing 2). Most of the electric models in the coupled energy system models use this interface, since it is sufficient for surveys that do not consider voltage stability, load flow calculations or non-symmetrical

three phase systems. There are two more electrical interfaces implemented, one adding voltage and reactive power and the other considering an adjustable number of phases. However, using these advanced electric interfaces to simulate electromagnetic phenomena with time constants in the range of milliseconds has proven to be unmanageable for coupled energy system simulations because of computational restrictions.

Listing 1. Interface for real fluids.

```
connector FluidPort "Fluid port in
  TransiEnt library"
  TILMedia.VLEFluidTypes.BaseVLEFluid
  Medium "Medium model";
flow Modelica.SIunits.MassFlowRate m_flow
  "Mass flow rate from the connection
  point into the component";
Modelica.SIunits.AbsolutePressure p "
  Thermodynamic pressure in the
  connection point";
stream Modelica.SIunits.SpecificEnthalpy
  h_outflow "Specific thermodynamic
  enthalpy close to the connection point
  if m_flow < 0";
stream Modelica.SIunits.MassFraction
  xi_outflow[Medium.nc-1] "Independent
  mixture mass fractions m_i/m close to
  the connection point if m_flow < 0";
end FluidPort;
```

Listing 2. Simple electrical interface.

```
connector ElectricPowerPort_L1 "General
  electrical interface in TransiEnt
  library"
flow Modelica.SIunits.ActivePower P "
  Active power in connector";
Modelica.SIunits.Frequency f "Frequency
  of grid";
end ElectricPowerPort_L1;
```

4 Example of Use: Transient Simulation of Coupled Energy Grids

In this section, an example for a coupled energy grid simulation will be given. Results from the simulated coupled energy system model are presented in order to illustrate the application possibilities of the library.

4.1 Description

The Dymola diagram of the system is shown in Figure 4. For consumer and renewable energy power plants static table-based models are used. The models of the electric power and storage plants are also table-based but consider transient effects. The model approach of the large-scale CHP plants is described in Section 3.4. The model

of the power-to-heat system is first order-based with an equation-based heat exchanger.

As an application example an assumed energy system of Hamburg in the year 2050 is simulated. In this context, energy system means electricity grid and district heating network (DHN). The general assumptions for this future scenario are:

- The electric and district heating demand profiles are the same as in 2012.
- The additional capacities of RE generators in 2050 are based on the expansion scenarios of the Renewable Energy Law (EEG) (Federal Republic of Germany, 2014).
- The structure of the conventional generation park is based on the 2012 generation park in Germany but without nuclear power which is removed without replacement.
- The generation park is scaled down using the peak load of Germany (82 GW) and Hamburg (2 GW), respectively.
- Wind and PV generation profiles (normalized by installed capacity) is the same as in 2012 and taken from 50Herz Transmission GmbH (2015).
- The district heating grid is fed by two black coal CHP plants (CHP Wedel in the west and CHP Tiefstack in the east of the city) and one natural gas peak load heating plant (Heating Plant Hafen in the center of the city).
- The distribution of electric power is not constrained by physical bottlenecks of the transmission or distribution networks.

Prior to the simulation in Dymola the unit commitment schedule of the conventional power park is generated. For this purpose a mixed-integer linear programming model has been derived that is explained in detail in Dubucq and Ackermann (2015) and is solved using the Matlab Optimization Toolbox (The MathWorks Inc., 2014). In essence, this approach allows to minimize the total operational cost of a given electric generation park subject to a list of physical and economic constraints. These are:

- The thermal and electricity demand has to be covered at every instant.
- All power plants are restricted by a minimum and maximum power level and a maximum power generation gradient.
- The pumped storage plant is additionally constrained by energy storage capacity.

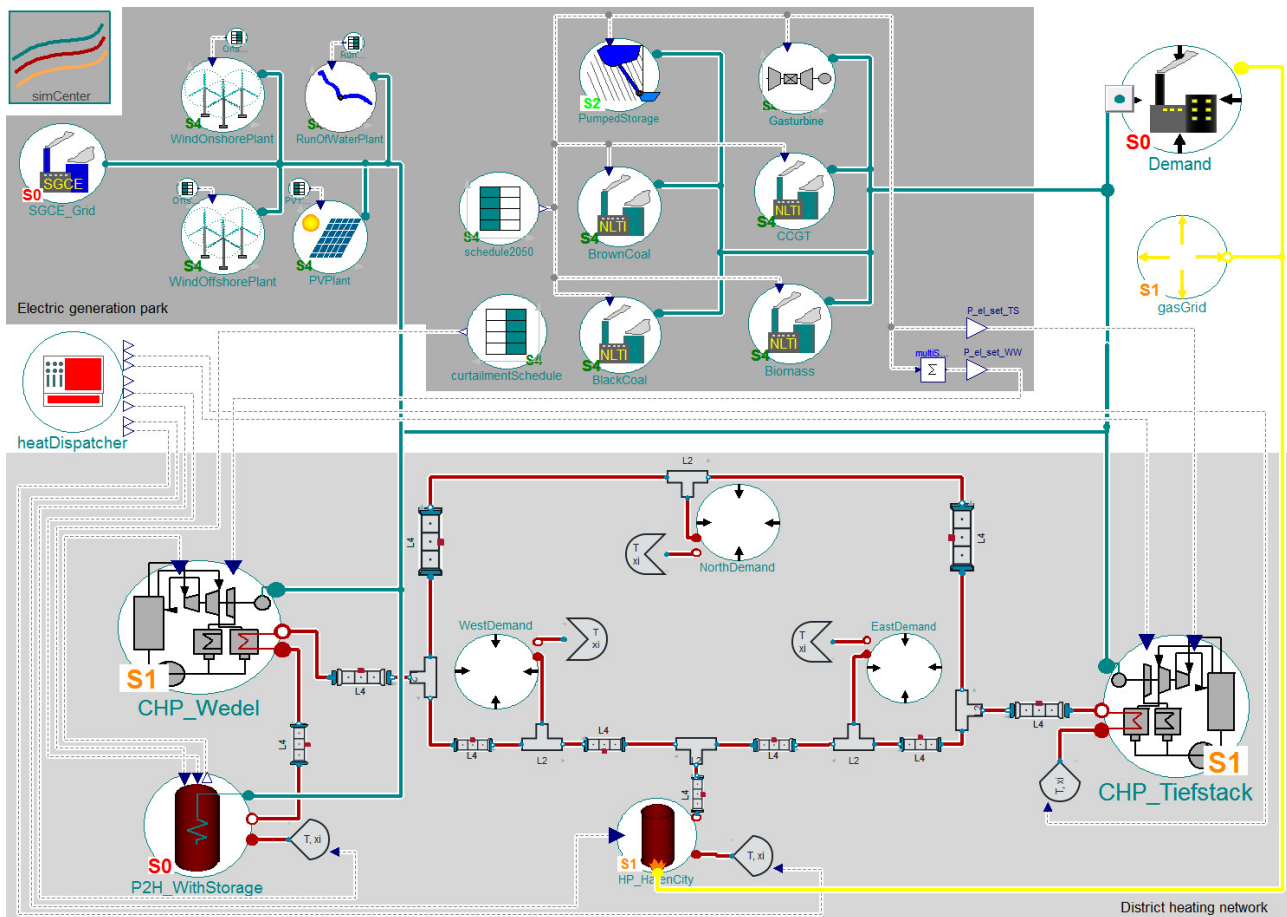


Figure 4. Dymola diagram of the simulated example for the coupled energy grids of Hamburg, showing the electricity generation park (top) and district heating network including the Power-to-Heat units (bottom).

- Minimum and maximum electric generation of CHP plants is depending on the current thermal heat generation.
- RE generation has feed-in priority.
- In every time step a symmetric electric reserve capacity of ± 142 MW has to be available to ensure grid stability.

4.2 Results

The resulting quarter hourly electricity production schedule is used in the coupled Modelica model (see Figure 4) where the simulation is carried out over an entire year with a variable step solver and the time resolution of the input table data is quarter hourly. All parameters for the considered generation park in this scenario can be found in Table 1.

Figure 5 shows the resulting weekly generated energy by renewable and must-run plants together with the weekly energy demand in 2050. The resulting share of renewable energies in the simulated year is 64 % whereas the targeted value by the German Renewable Energy Law for 2050 is 80 %. The reason for this small share

in relation to the targeted value is a large amount of RE generation that can not be integrated into the electric grid due to its fluctuating nature on the one hand and non-dispatchable generation units on the other hand. The non-dispatchable generation consists of three components: the first component is the above mentioned electric reserve capacity of ± 142 MW that must be available in order to compensate power imbalances and thus ensuring electric grid stability. Secondly, every CHP plant has a specific minimum electric power generation depending on the current thermal heat generation needed to match the district heating demand. The third component is the RE generation that has priority feed-in by the EEG (Federal Republic of Germany, 2014). However, as can be seen from Figure 5, the renewable generation cannot fully be used, due to the first two components of non-dispatchable generation and a lack of storage capacities. This leads to the operational necessity to curtail renewable energy generation in 3831 hours of the year which amounts to 2.39 TWh. Even though the yearly generated energy from non-dispatchable sources (13.3 TWh) is only 3 % higher than the electric energy consumption (12.9 TWh), a fossil generation of 4.66 TWh is needed to match the electric demand at all times.

Table 1. Parameters of the simulated electricity generation park.

Plant Type	Capacity	Efficiency	CO ₂ -Emissions	Minimum	Maximum Power	Var. Costs	Start-up Costs
	MW	%	Power g/kWh _{th,fuel}	Gradient %	%/min	€/MWh	€/MW
Brown Coal	574	34	403	40	6	22.7	81.8
Black Coal	688	40	337	30	8	27.9	81.8
CCGT	598	52	202	20	10	48.2	30.5
Gas Turbines	133	35	202	20	12	100.2	16.5
Pumped Storage	173		0	0	100	35.3	0
Biomass	322		0	0	100	38.3	0
Run-Off Water	123						
Photovoltaic	2446						
Wind Onshore	2753						
Wind Offshore	1327						
Sum	9137						

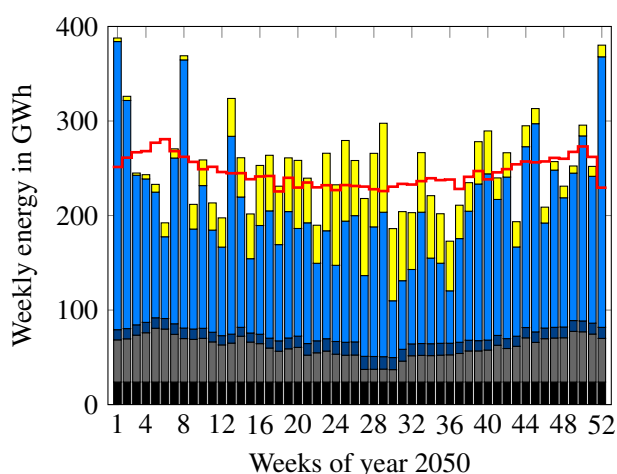


Figure 5. Weekly averaged energy consumption (—), and must-run generation: Spinning reserve (■), Minimum generation from CHP plants (■), Run-Off Water (■), Wind (■) and Photovoltaic (■) plants.

These results demonstrate the need for storage capacities and flexibility options to reach higher shares of RE generation. The objective of the developed library is to investigate measures resulting from a coupled system simulation approach that takes into account not only the electric but also the heat demand. One option following from this approach is to use the excess electricity to cover the heat demand (Power-to-Heat) and thereby using the district heating grid as a flexibility option. In fact, it has been proposed to install up to four 25 MW electric steam generators in the district heating grid of Hamburg together with a 1.9 GWh hot water storage in order to increase the flexibility of heat production in the western CHP plant (Erker, 2013). In order to assess the impact of such a measure another yearly simulation has been carried out in which such electric steam generators and a hot water storage have been added to the model. The results for electric and thermal power generation from this simulation together with the results from the base scenario can be found in Figure 6 for the first week of 2050 in

quarter hourly resolution. As can be seen from the illustrated week very large amounts (up to 2 GW) of excess generation occur in the year 2050 which can not be used in the electric grid due to lack of storage capacities. The 173 MW pumped storage plant can only shift 162 GWh of otherwise curtailed electric generation to periods of low renewable generation, whereas the total surplus energy amounts to 2.39 TWh during the entire year. While the renewable generation covers most of the demand, the must-run generation from the two CHP plants that cover the district heating grid demand is always present during the illustrated winter week. Despite the large excess generation from RE generators their non-dispatchable nature becomes clear visible in some periods (e.g. Monday morning and Friday noon), where dispatchable plants have to start up to cover demand in order to compensate for the decreasing wind energy offer.

The increased flexibility of the energy system introduced by the Power-to-Heat units leads to a use of 362.4 GWh of otherwise curtailed renewable energy (15.2 %) as can be seen in the bottom, right part of Figure 6. While the Power-to-Heat unit has only minor impact on the electric dispatch it does replace thermal generation from the black coal CHP plant by virtually emission free generation from RE sources. This effect can further be examined in Figure 7 where the mass of CO₂ emissions in both future scenarios and a 2012 base scenario are illustrated for the entire year. Comparing the results for the 2012 scenario and the future scenarios the effect of RE integration on CO₂ emissions becomes obvious with a total reduction of 5.9 million tons of CO₂ emissions from electricity generation which is equivalent to a 70.8 % reduction of emissions in the simulated energy system. This is also equivalent to a 80 % reduction with respect to emission from electricity in 1990 which is equal to the value stated by the German government coalition agreement (Federal Republic of Germany, 2013). However, it should be noted that this targeted value applies to the total CO₂ emissions in Germany (including emissions from transportation, etc.). The spe-

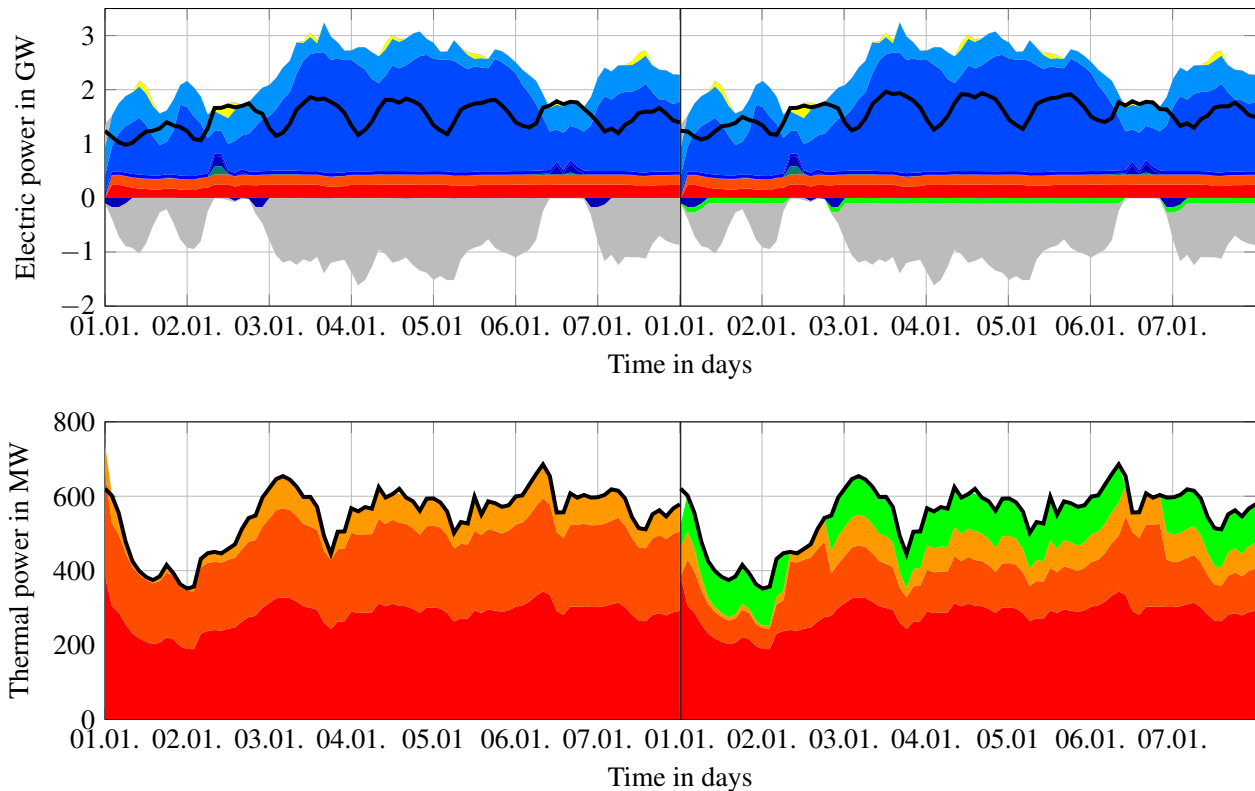


Figure 6. Simulated electric (top) and thermal (bottom) power generation in Hamburg for the first week (Sunday till Saturday) of 2050 scenario without Power-to-Heat (left) and with Power-to-Heat (right). ■ CHP Wedel ■ CHP Tiefstack ■ Biomass ■ Pumped Storage ■ Run-Off Water ■ Onshore Wind ■ Offshore Wind ■ Photovoltaic ■ Excess Electricity Generation ■ Heating Plant Hafen ■ Power-to-Heat — Electric Demand (top), Thermal Demand (bottom).

cific emission from electricity generation of the 2012 reference simulation amounts to 587 g/kWh whereas statistical evaluations in (Umweltbundesamt, 2015) give a slightly lower value of 562 g/kWh. This deviation is due to the replacement of nuclear plants that are emission-free.

The additionally used RE generation in the Power-to-Heat future scenario leads to a reduction of 51.2 thousand tons of CO₂ emissions. This is equivalent to a 8.3 % reduction of emissions from district heating grid operation and 1.7 % of total emissions in the simulated coupled system (electricity grid and DHN) with respect to the future scenario without Power-to-Heat. The specific emissions of district heating generation are reduced from 143 g/kWh to 131 g/kWh. The States of Germany working group of energy balances quantifies the specific DHN emissions of Hamburg in 2012 to 203 g/kWh (LAK-Energiebilanzen, 2015). One reason for this deviation is that the total production, hence total CO₂ mass flow emission, of the CHP plants is lower due to the increased RE electric generation. Another possible explanation is the method used for the allocation of CO₂ emissions to heat and electricity production from CHP plants. In the presented results the allocation is done using the simpli-

fied efficiency method described in (Mauch et al., 2010):

$$\dot{m}_{\text{CO}_2, \text{el}} = \dot{m}_{\text{CO}_2, \text{tot}} \cdot \frac{\eta_{\text{th}}}{\eta_{\text{th}} + \eta_{\text{el}}} \quad (1)$$

$$\dot{m}_{\text{CO}_2, \text{th}} = \dot{m}_{\text{CO}_2, \text{tot}} \cdot \frac{\eta_{\text{el}}}{\eta_{\text{th}} + \eta_{\text{el}}} \quad (2)$$

where $\dot{m}_{\text{CO}_2, \text{tot}}$ denotes the total CO₂ emissions and η_{th} and η_{el} are the thermal and electric efficiency of the CHP plant respectively. This method however, allocates a larger amount of emissions to the electrical side since the electrical efficiency is smaller than the thermal heat production efficiency which takes into consideration the fact that electricity is pure exergy and in this sense thermodynamically more valuable. The allocation method used in (LAK-Energiebilanzen, 2015) is the *alternative generation method* which leads to somewhat higher emissions on the thermal side (Mauch et al., 2010). In the final library different methods for CO₂ allocation will be made available to the user using replaceable models.

The time evolution of the CO₂ emissions from DHN operation is dependent on the season which is noticeable by the slope of the curve that is high during heating periods and low during the summer. This also applies to emissions from electricity generation because a decrease

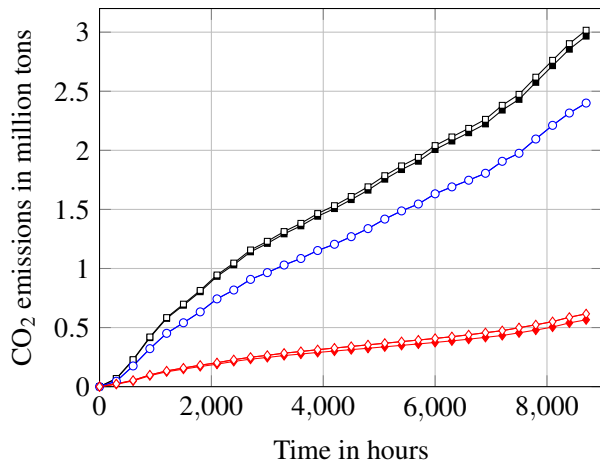


Figure 7. CO₂ emissions in future base scenario (empty symbols) and future scenario with Power-to-Heat units (filled symbols). Total emissions of electricity generation and DHN operation (—■—), emissions from electricity generation (—○—) and emissions from DHN thermal generation (—◇—).

in heat demand also leads to a decrease of must-run capacity (see also Figure 5).

5 Summary and Outlook

In this paper the development status of the TransiEnt library is presented. The library is being developed within the *TransiEnt.EE* research project. After project completion, the library will be made freely available under the terms of the Modelica license agreement. Although the library is still in the development phase, its current status already allows the simulation of RE integration scenarios.

In particular, the exemplary simulation showed that the curtailment of RE could be effectively reduced by the use of power-to-heat and heat storage units. Under the selected assumptions, 15.2 % of the originally curtailed energy could be integrated. This additional flexibility measure also leads to a reduction of the system's CO₂ emissions. Under the selected assumptions, the CO₂ emissions of the electricity generation and district heating operation together are reduced by 1.7 %. The CO₂ emissions of the district heating network operation are reduced by 8.3 %.

In the following project phase, the development team will focus on the detailed definition of the project scenarios, which include a central-oriented scenario, a decentralized scenario, a demand-side-management scenario and a storage scenario. In the mean time, the development of the library will continue. This development pursues the modeling of additional components and the improvement of existing ones, considering the requirements evolving from the scenarios.

6 Acknowledgements

The authors would like to acknowledge all supporters of the *TransiEnt.EE* research project, especially XRG Simulation GmbH and the project's advisory board. The project is funded by the German Federal Ministry for Economic Affairs and Energy on the basis of a decision by the German Bundestag (BMWi 03ET4003).

References

- 50Herz Transmission GmbH. Grid Data, 2015. URL <http://www.50hertz.com/en/Grid-Data>.
- BMWi. Ein Strommarkt für die Energiewende - Diskussionspapier des Bundesministeriums für Wirtschaft und Energie (Grünbuch). Technical report, Bundesministerium für Wirtschaft und Energie, 2014.
- Johannes Brunnemann, Friedrich Gottelt, Kai Wellner, Ala Renz, Andre Thüring, Volker Roeder, Christoph Hasenbein, Christian Schulze, Gerhard Schmitz, and Jörg Eiden. Status of claraccs: Modelling and simulation of coal-fired power plants with CO₂ capture. In *Proceedings of the 9th International Modelica Conference*, September 2012.
- Ernst Chudzienski. Hamburgs größte Wärmetransportleitung im Bau. *Fernwärme International*, 5(16):328–338, 1987.
- ClaRa Library, 2015. URL <http://www.claralib.com/>.
- Dassault Systemes. Dymola, 2012. URL <http://www.3ds.com/products-services/catia/products/dymola>.
- Pascal Dubucq and Günter Ackermann. Frequency Control in Coupled Energy Systems with High Penetration of Renewable Energies. In *International Conference on Clean Electric Power*, pages 336–342, Taormina, 2015. IEEE.
- ENTSO-E. European Network of Transmission System Operators for Electricity, 2015. URL <http://www.entsoe.eu>.
- Martin Erker. Innovationskraftwerk Wedel, 2013.
- European Commission. A Roadmap for moving to a competitive low carbon economy in 2050. Technical report, European Commission, 2011.
- Federal Republic of Germany. Coalition agreement 18th legislative period, 2013. URL <http://www.bundesregierung.de/Content/DE/StatischeSeiten/Breg/koalitionsvertrag-inhaltsverzeichnis.html>.
- Federal Republic of Germany. Gesetz für den Ausbau erneuerbarer Energien (Erneuerbare-Energien-Gesetz - EEG 2014), 2014. URL <http://www.bmwi.de/BMWi/Redaktion/PDF/G/gesetz-fuer-den-ausbau-erneuerbarer-energien,property=pdf,bereich=bmwi2012,sprache=de,rwb=true.pdf>.

Hamburg University of Technology. TransiEnt.EE Project, 2013-2016. URL <http://www.tu-harburg.de/transient-ee>.

Alfons Kather, Volker Röder, Christoph Hasenbein, Gerhard Schmitz, Kai Wellner, Friedrich Gottelt, and Lasse Nielsen. DYNCAP - Dynamische Untersuchung von Dampfkraftprozessen mit CO₂-Abtrennung zur Bereitstellung von Regelenergie. Final report, Bundesministerium für Wirtschaft und Energie, 2014.

LAK-Energiebilanzen. Spezifische CO₂-Emissionen der Strom- und Fernwärmeerzeugung in kg CO₂/GJ (Stand 20.07.2015), 2015. URL <http://www.lak-energiebilanzen.de/dseiten/co2BilanzenAktuelleErgebnisse.cfm>.

Wolfgang Mauch, Roger Corradini, Karin Wiesemeyer, and Marco Schwentzek. Allokationsmethoden für spezifische CO₂ - Emissionen von Strom und Wärme aus KWK-Anlagen. *Energiewirtschaftliche Tagesfragen*, 55(9):2-4, 2010.

The MathWorks Inc. Optimization Toolbox, 2014. URL <http://de.mathworks.com/products/optimization/>.

TLK Thermo GmbH. TILMedia Suite, 2015. URL <http://www.tlk-thermo.com/en/software-products/tilmedia.html>.

Umweltbundesamt. Entwicklung des CO₂-Emissionsfaktors für den Strommix in Deutschland in den Jahren 1990 bis 2012 (in Gramm pro Kilowattstunde), 2015. URL <http://de.statista.com/statistik/daten/studie/38897/umfrage/co2-emissionsfaktor-fuer-den-strommix-in-deutschland-seit-1990>.

Mathematical Model of Soot Blowing Influences in Dynamic Power Plant Modelling

C. Gierow¹ M. Hübel¹ J. Nocke¹ E. Hassel¹

¹Chair of Technical Thermodynamics, University of Rostock, Germany ,
{conrad.gierow,moritz.huebel,juergen.nocke,egon.hassel}@uni-rostock.de

Abstract

Due to the increasing integration of renewable energy sources in the existing power grid the conventional power plants have to set their focus more on flexibility and grid stabilization than supplying the base load. Since this task was not foreseeable when designing the currently existing power plants, they will have to suffer completely different load scenarios than expected. Dynamic modelling of complete steam cycles is a promising way to study the power plant operation of various future scenarios. To adapt the model to real power plant behaviour, especially with a focus on control events, the implementation of effects due to steam blown into the gasside part of the boiler in order to detach soot from the heating surfaces (soot blowing) seem to bring great efforts concerning model validity. Furthermore special control optimizations can be done, for example on spray injection at soot blowing events. In this study temperature measurement data is used in combination with a highly detailed boiler model of a 550 MW hard coal fired power plant to build a mathematical model of soot blowing influence on the different heat exchangers.

Keywords: Dynamic Modelling, Power Plant, Soot Blowing, Mathematical Modelling, ClaRa, Validation

1 Introduction

During normal operation of a power plant different chemical reactions lead to solid particles that are carried by the flue gas through the entire boiler. The amount and the composition of these particles mainly depend on two parameters. The first is the kind of fuel that is burned in the furnace. Using hard coal for example will lead to much less produced solid particles compared to burning waste and biomass as a substitute fuel. Secondly the arrangement and type of the burners influences the formation of the flame and thus influences the homogeneity of heat release and flame temperatures which might result in particle formation.

Parts of the produced amount of soot are taken up by the heat exchangers that are passed by the flue gas. This

mechanism is called fouling. As written in (Effenberger, 2000) there are different impacts on the furnace. The rate of heat transfer of the tube bundles decreases which affects the temperature field such that it increases towards the end of the boiler. Furthermore the reduction of the flue gas cross section leads to higher flue gas velocities. Overall the plant efficiency decreases with rising fouling of the heat exchange surfaces because of a higher flue gas pressure drop over the boiler and decreasing steam temperatures.

Due to this facts, the aim is to have a minimum fouling. Since frequent shut-downs of the entire plant to clean the heat exchangers are not desired, so called soot blowers are used to blow the attached particles from the heat exchangers. To avoid a cool down of the outer layer of the heat exchangers, superheated steam is used for this purpose.

2 Informative Background

This study has been carried out under the programme "THERRI" (Thermisches Ermüdungsrischwachstum - thermal fatigue crack growth) that is funded by the German Federal Ministry for Economic Affairs and Energy. The aim of this project is the development of a method and guideline for the fracture-mechanical assessment of thick-walled components in fossil-fueled power plants. The Chair of Technical Thermodynamics Rostock develops dynamic power plant models to provide thermodynamic boundary conditions, i.e. thermal and mechanical loads for subsequent fracture-mechanical tests and analyses. The dynamic model was developed within the software environment "Dymola" using the programming language "Modelica". The components used to build the model are largely from of the ClaRa library that is described in the following part.

2.1 Used Library - ClaRa

The ClaRa (Clausius-Rankine) is an open source library that has been developed under the programme "Dyncap" which was as well as "THERRI" funded by the German

Federal Ministry for Economic Affairs and Energy. It is written in Modelica modelling language. It allows the user to model highly detailed complete power plants with a strong focus on their dynamic behaviour (ClaRa, 2015; Brunnemann et al., 2012).

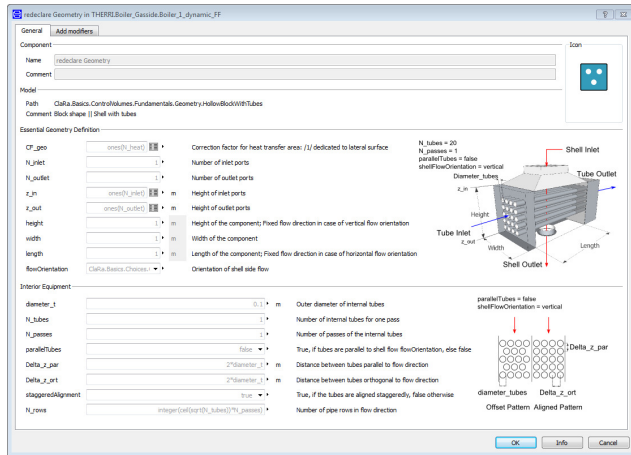


Figure 1. Example of Parameter GUI in ClaRa Library (ClaRa, 2015)

The library is divided into the following subpackages:

Table 1. Overview of the ClaRa library structure

	UsersGuide	modelling concept, contacts, license
	Examples	introducing examples
	Basics	base models and informational structure
	Components	component models, see Table 2
	SubSystems	definition of subsystems for large projects
	Visualisation	tools for visualisation, e.g. time plots
	StaticCycles	static models for calculation of initial values

The ClaRa is delivered with a package containing a set of stationary models which can be used to create a simplified, static and parameter-based mimic of the dynamic cycle. The result, a load depending set of parameters for mass flow, pressure and enthalpy for the complete cycle, can be used as initial guess values when linked to the respective parameters of the dynamic cycle. This allows the user to give flexible and consistent initial values at all dynamic components considering system topology and the possibility to use a cascaded initialisation with values

of upstream components for varying design points. The same procedure can be used to calculate nominal values for the main cycle.

The usable components are divided into base types containing various models at different levels of detail. Thus the user is able to create models that are as detailed as necessary and as simple as possible. Table 2 gives an overview of the implemented component classes. All

Table 2. Overview of the ClaRa subpackage *Components*

	BoundaryConditions	sinks and sources for water, steam and gas
	TurboMachines	fans, compressors pumps and turbines
	HeatExchangers	different heat exchanger types
	Mills	mills for preparation of solid fuels
	VolumesValvesFittings	volumes, valves and fittings for water, steam and gas
	MechanicalSeparation	gravitational phase separation, storage
	Furnace	base models for setting up entire boilers
	Electrical	electrical machinery
	Sensors	sensors for pressure, mass flow, temperature, etc.
	Control	base models for control purposes
	Adapters	adapters for related Modelica libraries
	FlueGasCleaning	denitrification, desulfurization and dedusting of flue-gas

component models are validated based on literature data and/or measurement data of existing components.

The ClaRa comes with a non-profit version of the TILMedia. Three different media types needed for the simulation of coal fired power plants are available. For pure mediums like water/steam there are table based and spline interpolated data available which are very encouraging concerning simulation speed and simulation stability, see (Schulze, 2013). The flue gas is described by a gas-vapour mixture similar to humid air. A mixture of

real fluids for application in CO₂-separation processes is supported too. For pressure loss and heat transfer the FluidDissipation library is used. For special purposes additional heat transfer correlations and radiation models can be used inside the combustion chamber.

For heat transfer the Modelica.Thermal connectors are used. The fluid connectors are in principle the same as the Modelica.Fluid connectors (despite the use of Temperature instead of enthalpy for gas flows) but they are using external substance properties and media types of the TILMedia ClaRa library making an own connector necessary. One main reason for the use of TILMedia ClaRa is that it is capable of calculating single and multicomponent vapor liquid equilibrium substances. The use of an external library comes with the advantage of a faster translation process and independence from the Modelica.Media being able to administrate, adapt and expand the needed substance properties according to the focus and requirements of the ClaRa.

2.2 Reference Power Plant

The investigated power plant is a hard-coal fired supercritical mono-block power plant Rostock. It is shown in Figure 2. The produced electric output is about 550 MW



Figure 2. Investigated reference power plant Rostock

with an overall efficiency of approximately 43.2% at full load. The maximum thermal output of the tower arranged forced circulation boiler is 1370 MW. At full load the plant operates with 417 kg/s feed water mass flow and a live steam pressure of 262 bar at 545°C. After depressurizing in the high-pressure turbine the steam is reheated to 562°C. In order to increase the fuel utilization ratio the plant is designed to decouple a maximum of 300 MW_{th} in combined heat and power cycle mode. Through this the utilization ratio can be risen up to 62%. A simplified schematic diagram is shown in Figure 3 in order to avoid an explanation of the entire layout of the plant.

3 Reduced Dynamic Model

There are two reasons in dynamic modelling why for detailed investigation on particular parts it is very useful to

cut out the most interesting part of the model and represent the left parts by boundary conditions. Firstly the duration of a simulation is heavily dependent on the deposited model size. Therefore a reduced model gives the ability to run the simulation more often e.g. for testing different sets of parameters. Secondly one get rid of the influences of uncertainties and assumptions made in the modelling of other devices and thus can compare different model results separately. For these reasons the complete plant has been reduced to the detailed boiler model that is described below. The model can be divided into 3 different parts, namely the gas side, water side and coal mill and combustion air section. The last part will not be treated in this paper, since its influence is not necessary for the investigations that will be made.

3.1 Gas Side

The gas side (see Figure 4a), which is responsible for modelling the heat release of the burned fuel, the transport of the flue gas through the entire boiler and the heat transfer via convection and radiation, is basically consisting the following submodels:

- Hopper and 4 Burner levels with coal dust inlet and an ideal combustion approach
- 2 Flamerrooms with a port for tertiary air input
- 4 Superheater levels containing tube bundles and carrier tubes
- 2 Reheater levels containing tube bundles and carrier tubes
- Economizer with finned tubes and carrier tubes

For each of these submodels the exact geometries of the surrounding walls and the tubes are implemented as well as the combustion formulas and different heat transfer correlations. The submodels are connected to their corresponding neighbours through flue gas and heat transfer ports. The latter ones are used to model the radiation between the different boiler levels. Besides that, each submodel has one heat transfer interface to model the heat transfer to the surrounding wall. Additionally the superheater, reheater and economizer submodels contain one heat port defining the interaction with the tube bundles and another one describing the heating of the carrier tubes.

As already mentioned, the regarded heat transfer mechanisms are convection and radiation. For the radiation part the formula to calculate the heat flow is based on the Stefan-Boltzmann law of radiation. In accordance with (VDI, 2006) it can be calculated according to equation (1),

$$\dot{Q}_{\text{rad}} = A_{\text{eff}} \sigma \frac{\varepsilon_{\text{W}}}{\alpha_{\text{G}} + \varepsilon_{\text{W}} - \alpha_{\text{G}} \varepsilon_{\text{W}}} (\alpha_{\text{G}} T_{\text{W}}^4 - \varepsilon_{\text{G}} T_{\text{G}}^4) \quad (1)$$

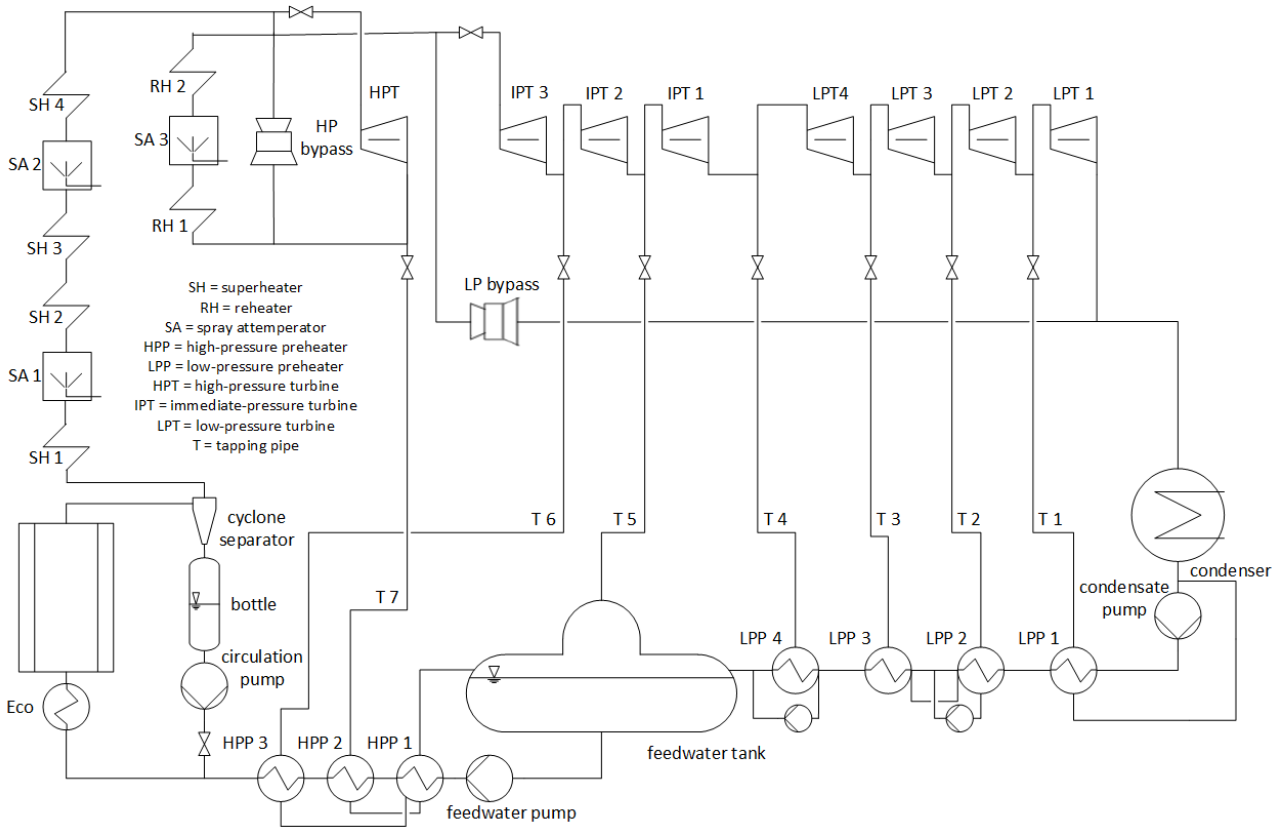


Figure 3. Schematic plot of the simplified water-steam cycle

where the emission and absorption coefficients of gas (ϵ_G , α_G) and the emission coefficient of the wall (ϵ_W) are approximated using the partial pressures of p_{CO_2} and p_{H_2O} and the equivalent thickness s_{gl} as well as the ash and coke load of the flue gas. T_G and T_W are the temperatures of the gas and the surrounding wall. σ stands for the Stefan-Boltzmann-Constant and the effective heat transfer area A_{eff} is defined as the actual heat transfer area multiplied by a fouling factor F_F which is used as the elementary connection between the physical and the developed mathematical model. Its behaviour will be further discussed in chapter 5.

The convective heat transfer is based on Newton's law with the same convention for A_{eff} and the Temperatures of the wall and the gas, T_w and T_g ,

$$\dot{Q}_{conv} = A_{eff} \alpha (T_w - T_g). \quad (2)$$

The heat transfer coefficient α is defined as

$$\alpha = \frac{Nu \lambda}{L} \quad (3)$$

where L denotes the characteristic length and Nu is a function of Re and Pr for laminar and turbulent flow.

3.2 Water / Solid Side

This submodel basically consists of pipe, valve and wall models. Each pipe has its assigned wall parametrized

with the necessary information for heat transfer and thermal inertia like inner and outer diameters (r_i , r_o), material properties (e.g. the coefficient of thermal conduction λ) and the total heat transfer surface. The conductive heat transfer through the walls is calculated based on (O'Kelly, 2012) with the Fourier law of heat conduction,

$$\dot{Q} = -2\pi\lambda l \frac{\Delta T}{\ln \frac{r_o}{r_i}} = \frac{\lambda A \Delta T}{r_o - r_i} \quad (4)$$

wherein ΔT denotes the temperature difference between the inner and the outer phase of the pipe wall and the effective heat conduction surface A follows equation (5).

$$A = 2\pi l \frac{r_o - r_i}{\ln \frac{r_o}{r_i}} \quad (5)$$

The pressure drop of the water/steam in the pipes is calculated with a nominal value Δp_{nom} and a linear mass flow dependency,

$$p_{in} - p_{out} = \Delta p_{nom} \frac{\dot{m}}{\dot{m}_{nom}}. \quad (6)$$

The convective heat transfer between the inner phase of the pipe and the fluid inside is assumed as well to be linear mass flow dependent with a nominal heat transfer coefficient. The different pipe diameters and geometries of the evaporator are regarded through different models for the hopper, burner, flame room and superheater

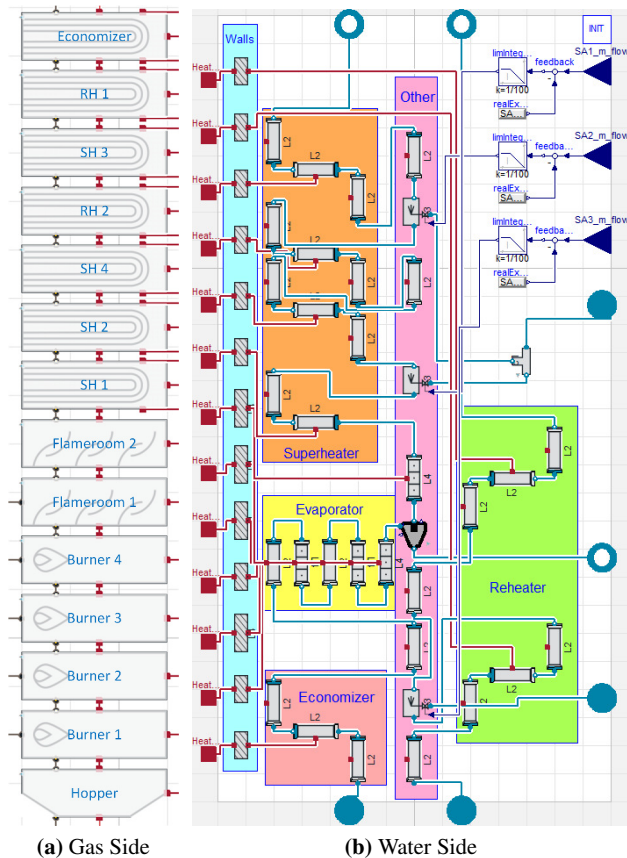


Figure 4. Gasside and Waterside parts of the developed boiler model

levels. Implementing this the relationship between the pipe lengths, the according heat transfer surfaces and the prevalent flue gas temperatures are taken into account. A schematic overview of this discretisation is given in Figure 4b. The arrowed connection lines are points where boundary conditions are applied. For controlling the steam temperatures in different levels, preheated water is sprayed into the steam pipe. The corresponding spray atomizers ("SA") are located between the superheaters 1 & 2, 3 & 4 as well as the reheaters 1 & 2. In the developed model the volume of main steam and spray is designed ideally stirred.

4 Analysis of Soot Blowing Influence

The soot blowing has complex physical effects on the heat transfer at the gas side part of the boiler. The attached constituents are partially blown away and chemical reactions may occur. For the applied 0D-/1D approach it appears to be sophisticated to design the physical background to model the exact behaviour. Additionally to the physical mechanisms the unit control is influenced as well. All these facts lead to the approach for this study, which is a hybrid method combining mathematical and physical modelling approaches.

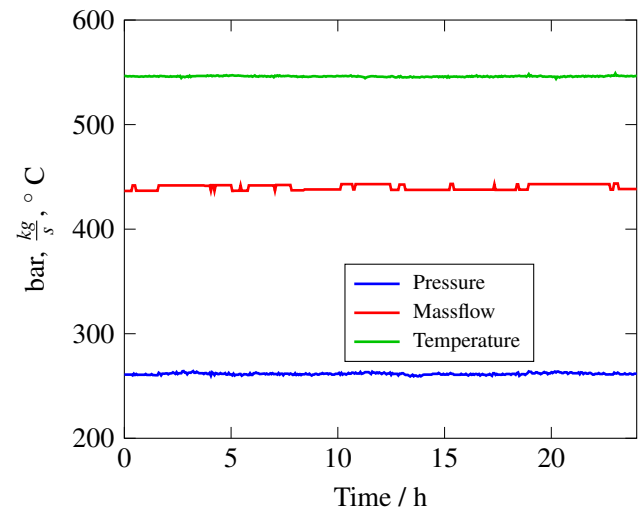


Figure 5. Live steam parameter of investigated load scenario

To identify the transfer functions, a real scenario of one day from the reference power plant (see Chapter 2.2) with steady load is used as database. This decision has been made due to the advantage, that in case of no load changes the soot blowing effects can be investigated separately without regarding other influences. The significant live steam parameters, as they can be seen in Figure 5, are nearly constant. Since the steam parameters do not change significantly the steam mass flowing through the whole boiler can be seen as stationary. This allows in further calculations to use the temperature difference between the inlet and the outlet of a superheater as an indicator for the heat flow.

In this study the focus of investigation lies on the first and fourth superheater. Nevertheless the method can be used to determine the soot blowing influence of further heat exchangers in the boiler. In Figure 6 the direct influence of soot blowing on one of the corresponding superheaters is shown. During the process the reached temperature difference rises and afterwards it decreases slowly until a certain normal level of $\Delta T \approx 12$ K in this example.

In addition to that the investigated process also affects the heat transfer of other heat exchanger areas. In this study the coherences between the heat transfer of superheater 1 and 4 are used as an exemplary case. Therefore Figure 7 shows the temperature difference of superheater 4 for the same scenario. In this plot, additionally to the former, the soot blowing intervals of some other heating surfaces are illustrated. It appears that the temperature difference decreases when blowing in superheater 1 level which is located approximately 4.5 m below the superheater 4. After these events it takes several hours to reach the previous amount of temperature difference or even a new operating point.

Another influence that can be seen is the soot blowing of the superheater 2 level that is located directly below the superheater 4. Having a look on Figure 7 it seems that the lance of superheater 2 also detaches some soot

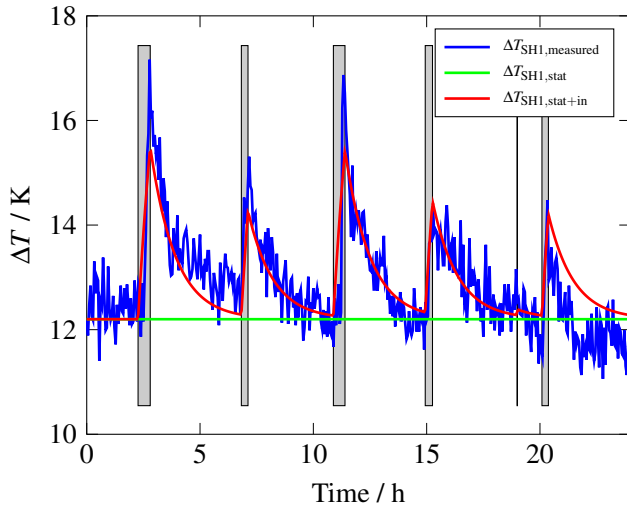


Figure 6. Steam temperature difference of superheater 1 with soot blowing interval (grey)

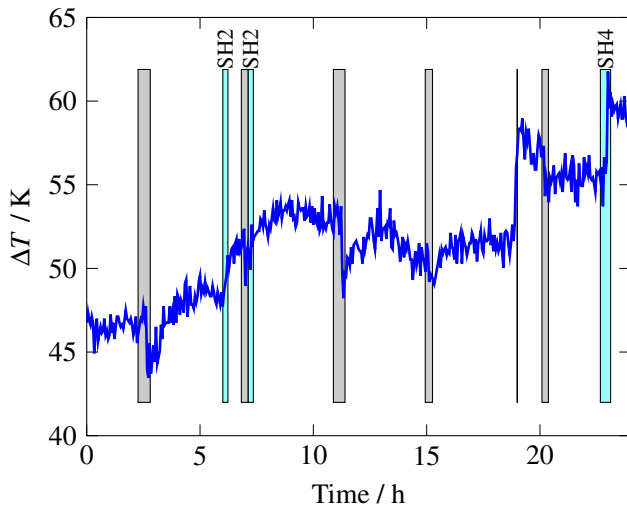


Figure 7. Steam temperature difference of superheater 4 with soot blowing interval of superheater 1 (grey) and superheater 2 and 4 (light blue)

and particles from superheater 4.

The steep ramp at 18 h is caused by a spray attemper-ator event and thus remains unregarded in this study.

5 Model Design

As already mentioned an exact physical approach is too complex to model in 0D/1D environments. Perhaps it would be worth to examine the behaviour using a detailed 3D-CFD method which is not part of this study.

The strategy of this study is to use a condensed mathematical model based on simple transfer functions. As described through the equations (1) and (2) in chapter 3.1 the heat flow of a superheater can be adjusted by the effective heat transfer area or more specifically the fouling factor F_F . This context is used to affect the different

tube bundles due to the soot blowing. To increase the overall heat transfer ability of a superheater, one raise the fouling factor of the according submodel at the gas side model and vice versa. Therefore the ClaRa implementation of F_F as a fixed parameter has been changed to a variable real value that is modifiable from outside the specific submodel. The method to identify the different values of the fouling factors is described in (Gierow et al., 2015) and thus is regarded as known for this study.

As an simplification for the modelling of the soot blowing influence, F_F is divided into two parts, $F_F = F_{stat} + F_{in}$. Herein F_{stat} describes the stationary or off-set part of the fouling factor at the current status. The corresponding temperature difference ΔT_{stat} is marked as a green line in Figure 6. The soot blowing is assumed to have no influence on this part. The second term is denoted as an influence factor, changing its value due to soot blowing events. Through this assumption it is possible to model and modify just the transient part of the fouling factor. In case of a stationary operation with no soot blowing it will remain at zero. This gives the huge advantage of the possibility to connect it to a binary signal indicating the on/off-state of the soot blowers. The entire mathematical model for this approach is described in the following.

As it may be seen by the red line in Figure 6, the transient temperature response is assumed to be proportional with a first order delay, also known as a PT_1 -element. Since all soot blowers in the boiler have to be taken into account, the complete system is defined with multiple inputs \underline{u} and outputs \underline{x} (MIMO). For m heat exchangers and n soot blowers the system is in the state-space representation form

$$\underbrace{\begin{bmatrix} \Delta \dot{T}_{in,11} \\ \Delta \dot{T}_{in,12} \\ \vdots \\ \Delta \dot{T}_{in,1n} \\ \Delta \dot{T}_{in,21} \\ \Delta \dot{T}_{in,22} \\ \vdots \\ \Delta \dot{T}_{in,mn} \end{bmatrix}}_{\dot{\underline{x}}} = \underline{S} \underbrace{\begin{bmatrix} \Delta T_{in,11} \\ \Delta T_{in,12} \\ \vdots \\ \Delta T_{in,1n} \\ \Delta T_{in,21} \\ \Delta T_{in,22} \\ \vdots \\ \Delta T_{in,mn} \end{bmatrix}}_{\underline{x}} + \underline{K} \underline{u} \quad (7)$$

where \underline{u} means a column vector of states of the investigated soot blowers. Active blowers are treated as $u_i = 1$ and inactive ones as $u_i = 0$. The state and input matrices are built as follows,

$$\underline{S} = \text{diag}(S_{11}, S_{12}, \dots, S_{1n}, S_{21}, S_{22}, \dots, S_{mn}) \quad (8)$$

$$\underline{K} = \begin{bmatrix} \text{diag}(K_{11}, K_{12}, \dots, K_{1n}) \\ \text{diag}(K_{21}, K_{22}, \dots, K_{2n}) \\ \vdots \\ \text{diag}(K_{m1}, K_{m2}, \dots, K_{mn}) \end{bmatrix} \quad (9)$$

The entries of the state matrix \underline{S} contain the time constants of the delay part of the PT₁ element in the form $S_{ij} = -\tau_{ij}^{-1}$. Since the fouling factors itself have no physical influence on each other, the non-diagonal entries of the matrix can be treated as zeros, $S_{ij} = 0$ for $i \neq j$. The input matrix \underline{K} represents the proportional gains between the soot blowers and the heat exchanger fouling factors. Since, considering the equations (1) and (2), the fouling factor influences the temperature difference in the first approximation linearly, the stationary temperature differences and fouling factors can be used to convert the state vector \underline{x} to the desired values of \underline{F}_{in} by multiplication with the output matrix \underline{C} ,

$$\begin{bmatrix} F_{in,1} \\ F_{in,2} \\ \vdots \\ F_{in,m} \end{bmatrix} = \underbrace{\text{diag}(c_1, c_2, \dots, c_m)}_{\underline{C}} \underline{x} \quad (10)$$

with

$$\underline{c}_i = \underbrace{[c_i \quad c_i \quad \dots \quad c_i]}_{n\text{-times}} \quad (11)$$

After the identification process the whole mathematical model of the soot blowing influence is implemented into the physical model of the gas side of the boiler as shown in Figure 8.

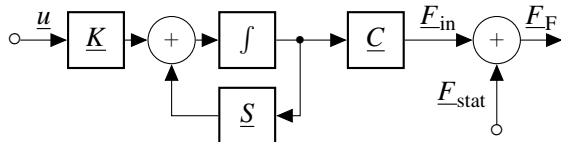


Figure 8. Block diagram of implementation in Dymola

6 Results and Validation

To avoid oversized matrices, the mathematical model is shrunk to the heat exchange of the superheaters 1 and 4 and the soot blowing in superheater 1 and 2 as already mentioned in chapter 4. Unfortunately the spray attenuator event at approx. 18h cannot be reproduced with the current status of the dynamic boiler model. For this reason the investigated time span for identification and validation is reduced by seven hours. During this time the soot blower of superheater 4 is not active and thus will be neglected from now on. Since the soot blower of the superheater 2 is located above the superheater 1, there is no influence between. For this reason the appropriate rows and columns in the matrices are omitted. This simplifies the model to the following form:

$$\begin{aligned} \dot{\underline{x}} = & - \begin{bmatrix} \tau_{SH1,1}^{-1} & 0 & 0 \\ 0 & \tau_{SH4,1}^{-1} & 0 \\ 0 & 0 & \tau_{SH4,2}^{-1} \end{bmatrix} \underbrace{\begin{bmatrix} \Delta T_{in,SH1,1} \\ \Delta T_{in,SH4,1} \\ \Delta T_{in,SH4,2} \end{bmatrix}}_{\underline{x}} + \\ & + \begin{bmatrix} K_{SH1,1} & 0 \\ K_{SH4,1} & 0 \\ 0 & K_{SH4,2} \end{bmatrix} \begin{bmatrix} u_{SH1} \\ u_{SH2} \end{bmatrix} \\ \begin{bmatrix} F_{in,SH1} \\ F_{in,SH4} \end{bmatrix} = & \begin{bmatrix} \frac{F_{stat,SH1}}{\Delta T_{stat,SH1}} & 0 & 0 \\ 0 & \frac{F_{stat,SH4}}{\Delta T_{stat,SH4}} & \frac{F_{stat,SH4}}{\Delta T_{stat,SH4}} \end{bmatrix} \underline{x} \end{aligned} \quad (12)$$

The identification problem thus is reduced to 6 independent variables. For handling this problem it is split into two sub-problems and then solved with the system identification toolbox of MATLAB[®].

Table 3. Identified parameters

Variable	SH1	SH4
ΔT_{stat} [K]	12.2	50.2
F_{stat}	0.64	0.61
$c = \frac{F_{stat}}{\Delta T_{stat}}$ [$\frac{1}{K}$]	0.05246	0.01216
τ_1 [s]	3992	5437
τ_2 [s]	-	4.5e9
K_1	2.154e-3	-2.364e-3
K_2	-	4.22e-3

For the investigated scenario the parameters in Table 3 have been identified. There are different aspects that can be deduced from some parameters. For instance the soot blowing at the superheater 1 has a negative gain for the temperature difference of superheater 4 and thus will reduce the temperature difference over this tube bank. Another point is the high delay time τ_2 of superheater 4. It appears that it is so high, that the delay part is negligible. This means the influence has only an integrative character.

Figure 9 and Figure 10 show the results of the already mentioned scenario simulated using the highly detailed boiler model described in chapter 3 with an implemented soot blowing model containing the parameters shown in Table 3. Both temperature trajectories can be reproduced using the developed model approach. The deviation after 12h in Figure 10 is reasoned by the rough discretisation of the measurement data which is used for the boundary conditions during the simulation. Based on the overall results the applied approach assuming a first-order behaviour appears to be sufficient.

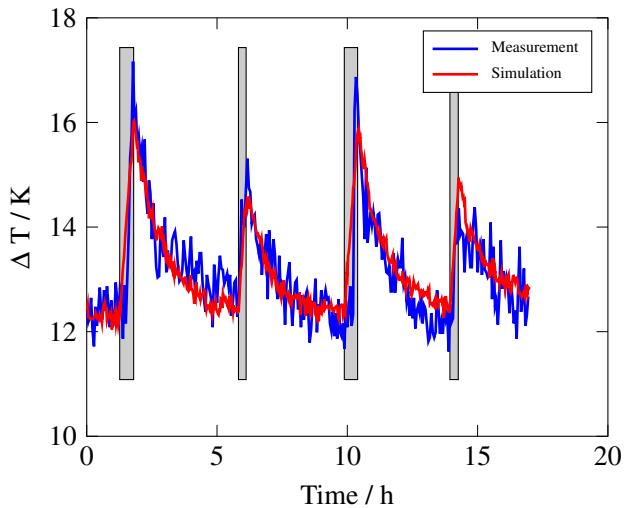


Figure 9. Comparison of simulated and measured steam temperature difference of superheater 1 with the corresponding soot blowing interval (grey)

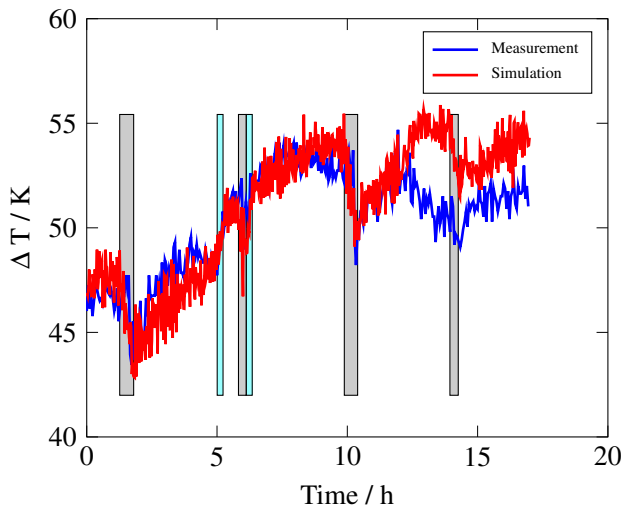


Figure 10. Comparison of simulated and measured steam temperature difference of superheater 4 with soot blowing interval of superheater 1 (grey) and superheater 2 (light blue)

7 Summary and Outlook

In this paper an innovative and in terms of computational costs resource-efficient algorithm to include the soot blowing influence into a dynamic power plant has been presented. The model allows to investigate both, the direct influence of soot blowing on the appropriate superheater and how it affects the superheaters that coming afterwards in flue gas direction. For a steady load scenario the accuracy and validity of the method has been shown.

In the next stages further studies will follow concerning load changes and scenarios with different stationary fouling factors. Furthermore this study could be compared to a more physical approach considering chemical reactions and the enthalpy flow into the flue gas. In

case of completely validated models, control optimisations could be possible, e.g. to avoid any influence of soot blowing on the spray injection dynamics.

8 Acknowledgement

The authors would like to thank the German Federal Ministry for Economic Affairs and Energy for funding the project. Furthermore we would like to thank all the partners in the project, especially the people working at the reference power plant for supporting us incessantly and giving us all the necessary information and data for the dynamic modelling. In addition we would like to thank the development team of the ClaRa and TILMedia ClaRa libraries from XRG Simulation GmbH and TLK-Thermo GmbH for their competent and fast support.

References

- Johannes Brunnemann, Friedrich Gottelt, Kai Wellner, Ala Renz, André Thüring, Volker Roeder, Christoph Hasenbein, Christian Schulze, Gerhard Schmitz, and Jörg Eiden. Status of ClaRaCCS: Modelling and simulation of coal-fired power plants with CO₂ capture. In *Proceedings of the 9th International Modelica Conference*, 2012.
- ClaRa. dyncap | ClaRa - Simulation von Clausius-Rankine-Kreisläufen, May 2015. URL www.claralib.com.
- Helmut Effenberger. *Dampferzeugung*. Springer, 2000. ISBN 3-540-64175-0.
- Conrad Gierow, Moritz Hübel, Jürgen Nocke, and Egon Hasel. Vergleich von Algorithmen zur Identifikation der Heizflächenverschmutzung. In *In Print: Kraftwerkstechnisches Kolloquium*, Dresden, 2015.
- P. O'Kelly. *Computer Simulation of Thermal Plant Operations*. Springer New York, 2012. ISBN 978-1-461-44256-1.
- Christian Schulze. Numerisch effizientes Modellieren von thermodynamischen Systemen. In *16. ITI Symposium*, Dresden, 2013.
- VDI. *VDI-Wärmeatlas*. Springer, Berlin, Heidelberg, 10th edition, 2006. ISBN 978-3-540-25503-1.

Flexibilization of coal-fired power plants by Dynamic Simulation

Marcel Richter¹ Florian Möllenbruck¹ Andreas Starinski¹ Gerd Oeljeklaus¹ Klaus Görner¹

¹Chair of Environmental Process Engineering and Plant Design, University of Duisburg-Essen, Germany
{marcel.richter, florian.moellenbruck}@uni-due.de

Abstract

Due to the strong expansion of renewable energies, the economical and technological boundary conditions for coal-fired power plants in Germany changed significantly over the last few years. Nowadays the flexibility in power production becomes increasingly important. This increasing flexibility requirement is caused by a more and more volatile residual load through the fluctuating power output from weather-dependent renewable energies such as wind power and photovoltaics. A similar trend can be observed in other European countries and even world-wide, where the expansion of renewable energies is pursued to reduce the emission of carbon dioxide. Dynamic simulation models play a central role in improving the flexibility of power plants as they offer a tool for the evaluation and improvement of the resulting highly transient operation. This paper presents the dynamic modeling of a coal-fired power plant in Modelica/Dymola using the power plant library *ClaRa* (Clausius-Rankine). The focus is on the detailed non-steady-state modeling of the steam generator and the validation of the dynamic simulation model. Additionally, first results of simulation studies about the integration of a thermal energy storage and the increase of the load change rate are presented.

Keywords: thermodynamics, dynamic simulation, steam power plant, flexible power plant, steam generator, validation, thermal energy storage, load change rate

1 Introduction

The share of renewable energies in power production in Germany increased significantly during the last years. This development has been supported by the German Renewable Energy Sources Act (EEG) which guarantees preferred feed-in into the electrical grid for renewable energies. The development of the power generation capacity in Germany between 2002 and 2012, shown in Table 1, illustrates the expansion of the renewable energies. The substantial growth of the total installed capacity from 127.0 to 184.4 GW is mostly covered through the renewable energies wind and sun (photovoltaics). The share of renewable energies in power production in Germany increased from 7 % in

2001 to nearly 25 % in 2013 [1]. According to the latest version of the EEG from August 2014 this development is to be continued towards the targets of 40 - 45 % in 2024, 50 - 60 % in 2034 and 80 % in 2050 [6].

Table 1: Power generation capacity in Germany in GW_{el} based on energy data [5] for 2002 and 2012 and on the assumptions of the Netzentwicklungsplan* (grid development plan) for 2024 and 2034 (scenario B) [7]

in GW_{el}	2002	2012	2024*	2034*
Hard coal	30.1	29.8	25.8	18.4
Lignite	21.6	24.2	15.4	11.3
Fuel oil	5.3	4.2	1.8	1.1
Gas	20.3	26.4	28.2	37.5
Nuclear	23.6	12.7	0	0
Water	8.9	10.4	14.7	15.7
Wind	12.0	31.3	67	97.3
Photovoltaic	0.3	32.6	56	59.5
Biomass	0.8	6.2	8.7	9.2
Other	4.1	6.6	5.2	5.0
Conventional	104.3	102.3	84.9	81.7
Renewable	22.7	82.1	138.6	173.3
Total	127.0	184.4	223.5	255

The requirements for a stable grid operation rise due to this growing share of renewable energies and the volatile character of their weather-dependent power production. The transformation of the electricity system leads to fundamental structural changes in the residual load, defined as the demanded power (load) minus a proportion of fluctuating power (e.g. from wind and sun).

Figure 1 shows the distribution of the residual load in the year 2012 as well as predicted values for 2024 and 2034. There will be periods in the future where the load is fully covered by the supply of the renewable energies. This leads to the appearance of negative residual loads. The shift of the residual load distribution towards lower values implicates the necessity for fossil fueled power plants to run at the lowest possible minimum load. Additionally, the requirement for fast and economic start-up and shut-down procedures can be derived from this figure, because conventional power plants have to perform

these procedures more often. A further aspect is that the maximum residual load in 2034 is still in a similar dimension in comparison to today. To ensure security of supply during periods of high residual loads as well, controllable steam power plant capacity has to be available – as long as not enough storage capacity (e.g. pumped storages, power2gas, etc.) is built-up to compensate the fluctuating power production of renewable energies.

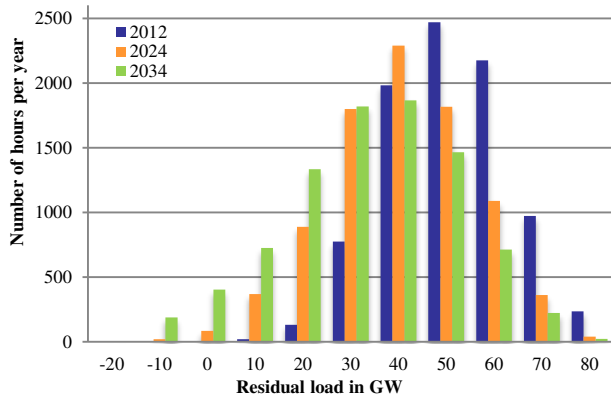


Figure 1. Histogram of the residual load in Germany (width of the histogram bars is ± 5 GW) [8]

Figure 2 shows the histogram of the residual load change between consecutive hours. The diagram illustrates a shift towards higher residual load changes for the future, both for negative and positive directions.

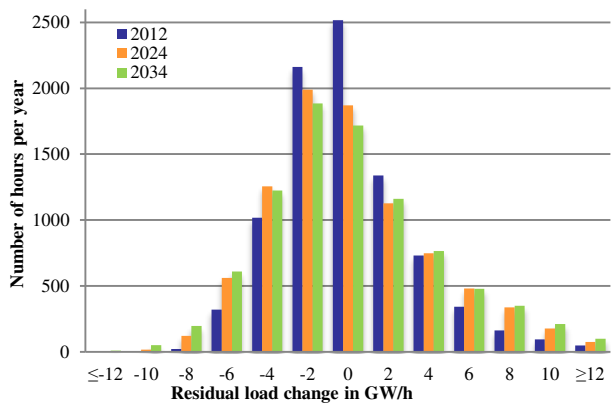


Figure 2. Histogram of the residual load changes between consecutive hours (width of the histogram bars is ± 1 GW/h) [8]

The rising volatility of the residual load results in increasing flexibility requirements to fossil fueled power plants regarding the load change rate, the start-up and shut-down procedures as well as the supply of control energy.

As a consequence of this development, the flexibilization of conventional power plants is one of the key challenges for the next years. This flexibilization is necessary to enable further integration of renewable energies while ensuring an economic operation of conventional steam power plants which ensure security of supply.

2 Applications of dynamic power plant simulation

Taking into account the future flexibility requirements to conventional power plants, the importance of dynamic power plant simulation increases. The dynamic power plant simulation offers a tool to model and calculate the transient operational behavior of existing or planned power plants.

The use of dynamic simulation models enables investigations about the following improvements in the power plant process [10]:

- Reduction of the minimum load
- Increase of the load change rate
- Reduction of the start-up and shut-down time
- Evaluation of the supply of control energy
- Evaluation of (thermal) energy storage concepts
- Evaluation of process quality during transient power plant operation

3 Simulation Software

Clara (Clausius-Rankine) is a free of charge and open source library of power plant components written in the modeling language Modelica. The library allows modeling and simulation of coal-fired power plants as well as heat recovery power plants, giving deep insight into their dynamic behavior [2], [3]. Both once-through and circulation boilers are supported. The library is structured component-wise including models for pumps, fans, turbines, heat exchangers, furnaces, electric motors, mills, valves, piping and fittings, as well as storage tanks and flue gas cleaning units. The library provides component models at different levels of detail supporting the user in creating system models tailored to their specific needs. The advantage of this concept is that the physical precision of a complex power plant model can be adapted to cope with the given simulation task without an unnecessary excess of computing time. The models are validated against literature and/or measurement data of existing plants, exemplified shown for the results of a pump in figure 3.

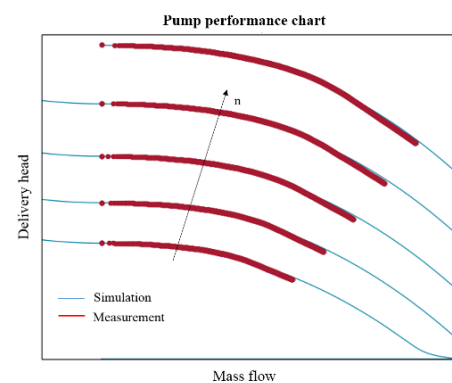


Figure 3. Validation results of a pump model

The library uses robust and fast media data from the *TILMedia* library as well as functions for pressure loss and heat transfer from the *FluidDissipation* library. In addition, special heat transfer correlations and radiation models are available for the flue gas path inside of a steam generator.

4 Modeling of steam power plants

Figure 4 shows the water-steam cycle of the modeled coal-fired power plant. This process flow diagram illustrates the three turbine groups (high, intermediate and low pressure turbine), the condenser, the feed water tank, the four low pressure and two high pressure preheaters and the subcomponents of the steam generator. The steam generator, as exemplarily shown later in figure 5, is an once-through boiler.

4.1 Procedure of model build-up

For dynamic simulations, the components of a power plant can be divided into steady and non-steady. The distinguishing criterion is the change rate of the component answering to a change in the thermodynamic boundary conditions (e.g. a change in temperature) reaching a new state of equilibrium. Table 2 shows the classification of the power plant components into steady and non-steady for the dynamic simulation model presented in this paper. Steady components have significantly smaller time constants in comparison to the non-steady components which means low influence to the dynamic behavior.

Table 2: Classification of power plant components into steady and non-steady based on [4]

Steady components	Non-steady components
Steam turbine	Heat exchanger
Pump	Steam pipe
Valve	Mixing point
Compressor	Feed water tank
..	Coal mill
	..

Based on this distinction, the procedure during the build-up of the dynamic simulation model can be divided into three steps, taking into account the difference between a steady-state simulation and a dynamic simulation. In the first step, a stationary simulation model with a detailed consideration of the steam generator (each heating surface is modeled and calculated) is developed using Epsilon@Professional. The stationary simulation model provides the starting values for the parameterization and initialization of the dynamic simulation model, which is built-up in the second step using the power plant library *Clara* in Modelica/Dymola. The dynamic simulation model consists of the combination of steady and non-steady components based on table 2 and a detailed consideration of the implemented control structures.

In the third step, the dynamic simulation model is validated by comparing the calculated values with measured operating data of the underlying power plant. After successfully completing this step, the dynamic simulation model offers the possibility to perform investigations to improve the flexibility of the regarded power plant process.

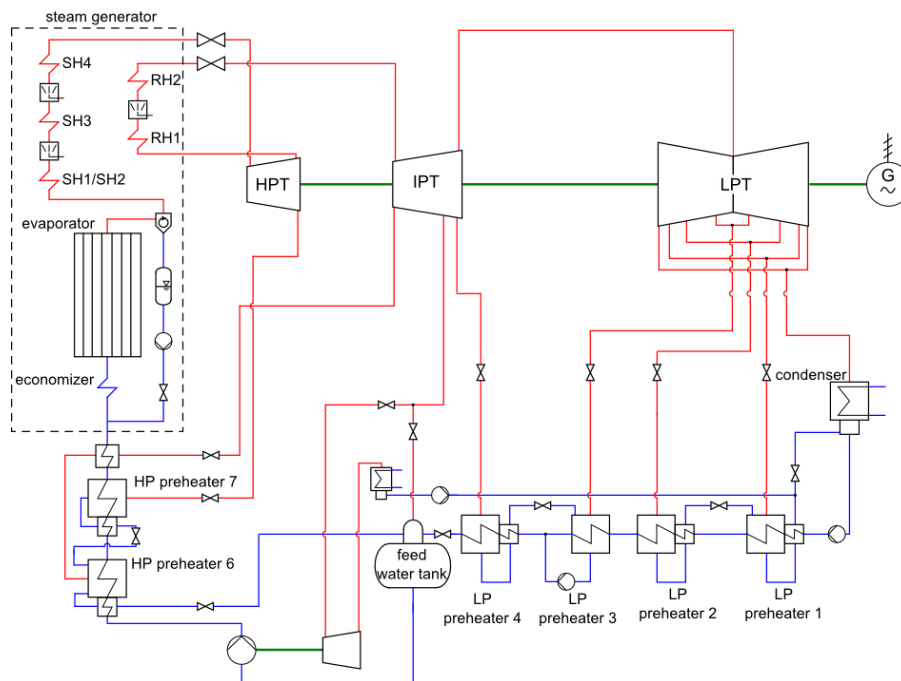


Figure 4: Process flow diagram of the modeled power plant

4.2 Detailed modeling of a steam generator

The structure of a detailed steam generator model within the *Clara* library is modular. The steam generator model itself can be divided into four areas:

1. Coal preparation and distribution
2. Flue gas path
3. Heating surfaces (walls and tube bundles)
4. Water flowing through the steam generator pipes



Figure 5. Exemplarily drawing of a steam generator [9]

Figure 6 illustrates the modular structure of the steam generator model in the *Clara* library. On the left side, the vertically discretized flue gas path is shown. The flue gas path is divided into several volume elements for every heat transfer section. Each component of the flue gas path can combine different functionalities, such as burner and/or heat transfer surface. Every component provides a model for the geometry, the velocity of coal and ash particles and the heat transfer correlation (e.g. radiative heat transfer within the flame room and/or convective heat transfer within the tube bundle heating surfaces).

The combustion chamber with the burners is represented by the component “Evaporator” at the starting point of the flue gas path. For the distribution of coal and combustion air, this component is connected with the coal and combustion air mass flow rates. The time-dependent behavior of the coal mills is represented by transfer functions. In flow direction of the flue gas the sections of a flame room (SH1) and the

tube bundle heat exchangers (SH2, SH4, RH2, SH3, RH1, ECO) are following. Each component in the flue gas path has a connector (red lines in figure 6) for the heat flow to the pipe wall. The components representing the sections for tube bundle heat exchangers have two additional connectors for the heat transfer to carrier tubes and tube bundle.

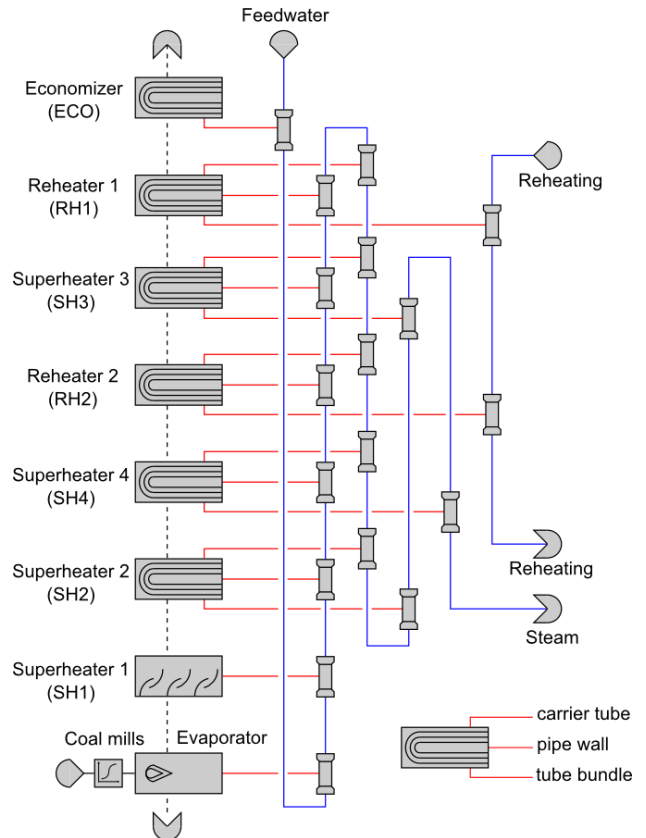


Figure 6. Modular structure of the steam generator

The material of the steam generator pipes is considered through a thin wall between flue gas and water-steam cycle, as exemplarily shown in figure 7. The tube-elements representing the water-steam side can be discretized by the user into several volume elements.

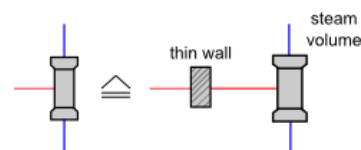


Figure 7. Wall-concept in Clara

The feed water initially flows through the economizer. This component is located at the end of the flue gas path and is connected in countercurrent. From the economizer the water flows through the downpipe drains to the evaporator. The following pipe wall (SH1) is arranged vertically and is modeled as a flame room. Subsequently, the steam flows through the three superheater surfaces of SH2, SH3 and SH4. The reheating takes place in the two reheater surfaces (RH1/RH2). Injection coolers are located between the surfaces of SH2/SH3, SH3/SH4 and RH1/RH2.

4.3 Control System

Besides the dynamic modeling of the power plant process, the control system has to be considered during the build-up of a dynamic simulation model. The control structures implemented within the control system have a major influence on the transient behavior of a power plant and consequentially also on the results of a dynamic power plant model. The goal of the power plant control system is the coordination of the interaction between boiler, turbine and generator to ensure efficient and safe operation of the power unit. The control system calculates setpoints and manipulated variables for the transient operation of a power plant. The entire control system has a high complexity and consists of a plurality of subordinate control loops for the command and control of various components and auxiliary units. The following control structures are implemented in the presented dynamic simulation model to achieve sufficient accuracy of the simulation results:

- Unit control
- Feed water control
- Steam temperature control
- Control of coal mills & fresh air

The main task of the unit control, shown in simplified form in figure 8, is to adapt the actual power output to the required target power output [12].

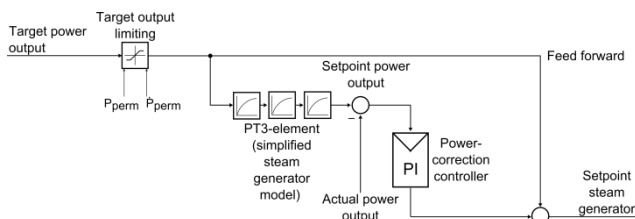


Figure 8. Simplified structure of the unit control implemented in the dynamic simulation model

To comply with the plant limits, the target output limiting ensures that the setpoint power output does not exceed the maximum permissible output P_{perm} and maximum permissible rate of change \dot{P}_{perm} . A step change of the target power output is thereby transferred into a straight line with the permissible rate of change \dot{P}_{perm} (in MW/min) as the gradient. The time behavior of the steam generator (especially coal mills and heat transfer) is represented by a PT₃-element. Following this, the difference between setpoint and actual power output is given to the power correction controller which determines an appropriate correction factor.

Figure 9 illustrates the simplified schematic structure of the steam temperature control. The regulation of the live steam temperature to the setpoint is achieved through the injection of cool water in front of the surfaces of SH3 and SH4. The mass flow of water to the injection coolers is controlled by a cascade connection of two PI controllers [11].

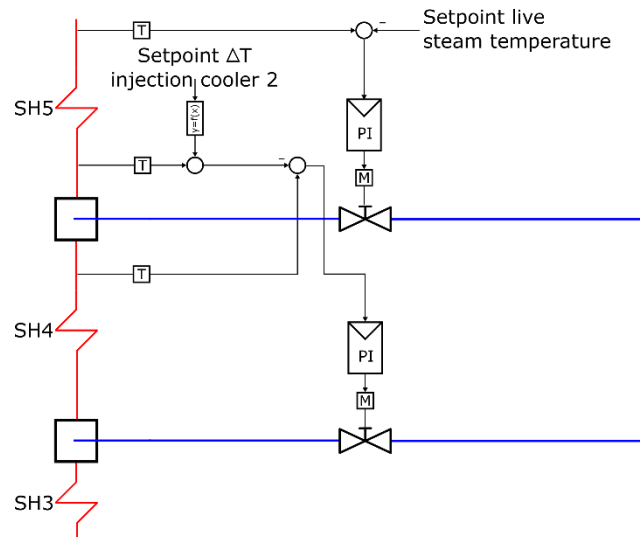


Figure 9. Simplified structure of the steam temperature control implemented in the dynamic simulation model

5 Validation against operating data

In order to prove the validity of the dynamic simulation model, measurement data from the underlying power plant are compared to the simulation results. The regarded load profile is characterized by several positive and negative load changes between minimum load and nearly full load.

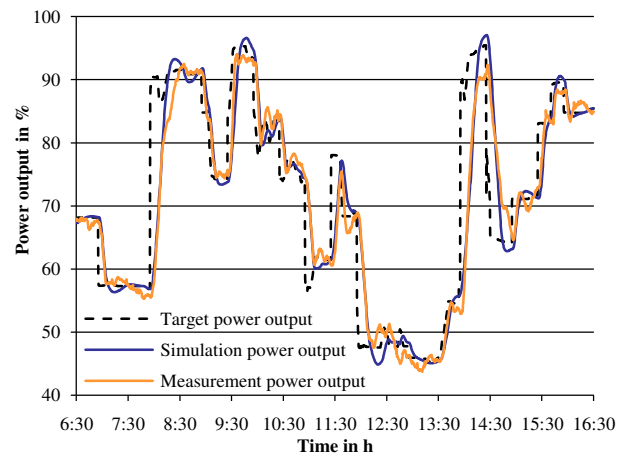


Figure 10. Comparison of the simulated power output (blue line) and measured values (orange line)

As described above, the power output is mainly controlled by the unit control with the target power output as the input variable (black dashed line). The results of the dynamic simulation model (blue line) show a high level of accordance to the measurement data (orange line) concerning the power output during the load profile. In particular, the rate of change during the load change processes is very well reproduced by the dynamic simulation model. This is due to the correct setting of the maximum permissible rate of change \dot{P}_{perm} within the unit control, as presented in the previous section.

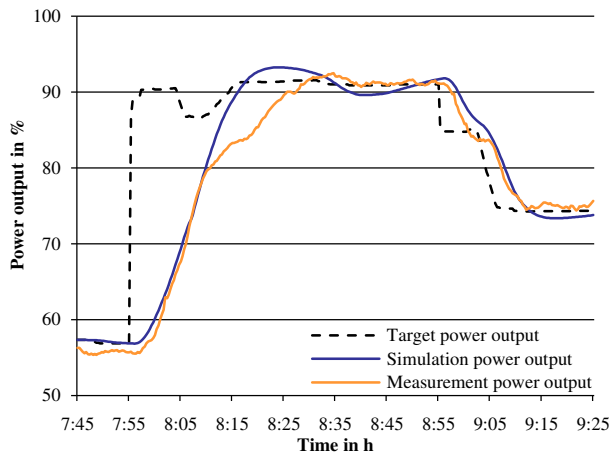


Figure 11. Comparison of the simulated power output (blue line) and measured values (orange line) between 7:45 and 9:25

Figure 11 shows a shorter section of this validation over a period of 100 minutes. In the time range between 8:15 and 8:35 a noticeable deviation has to be recognized between simulation and measurement. Due to a lower rate of change during this time range, the measurement data needs approximately ten minutes more to reach the target power output. This deviation can be explained by the fact, that the unit control implemented in the dynamic simulation model is a simplified structure in comparison to the highly complex structure of the underlying reference power plant. Here the authors see further development opportunities of the dynamic simulation model. Nevertheless, figure 11 shows that the power plant's dead and balancing times are simulated in a sufficient accuracy. Thereby, the dynamic simulation model is valid for the simulation studies about the integration of a thermal energy storage and the increase of the load change rate as presented later in section 5.

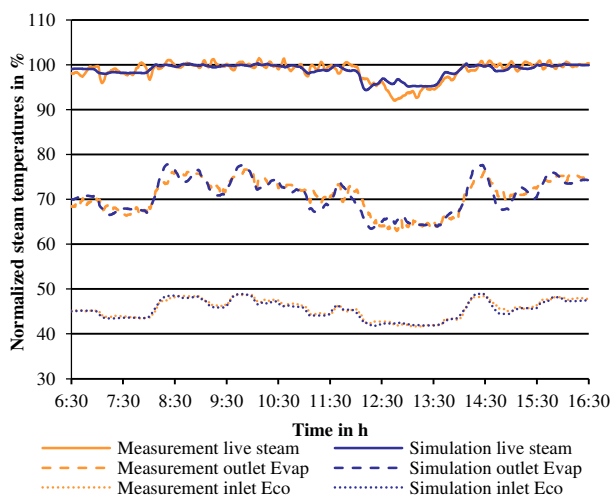


Figure 12: Comparison of simulated (blue) and measured (orange) water-steam temperatures

Figure 12 shows the comparison of simulated (blue) and measured (orange) water-steam temperatures to

prove the validity of the dynamic steam generator model. The diagram is normalized to the set value of the live steam temperature. The economizer inlet temperature, the evaporator outlet temperature and the live steam temperature show a good match between simulated and measured values during the load profile. Thereby, the detailed dynamic simulation model of the steam generator - as presented in section 4.2 - is proven successfully what can be claimed as a considerable achievement. Furthermore, the control of the live steam temperature to the set point confirms that the implemented steam temperature control is working sufficiently.

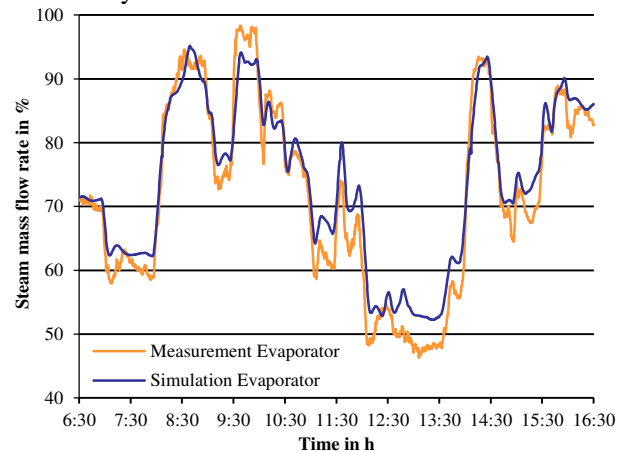


Figure 13: Comparison of simulated and measured mass flow of the evaporator

Finally, the time curve of the simulated evaporator mass flow, illustrated in figure 13, also shows a sufficient accuracy during the load profile.

Summing up, the comparison of simulated and measured values shows a good accordance in different load points and also during load changes. Thereby the validity of the dynamic simulation model is proven successfully and the model enables further investigations to increase the flexibility of the power plant process.

6 First results of the dynamic simulation model concerning flexibilization

6.1 Integration of a thermal energy storage

The integration of a thermal energy storage is one possible flexibilization measure that can be evaluated by the dynamic simulation model. A thermal energy storage can have different effects on the flexibility, depending on concept, point of integration and capacity. A thermal energy storage can be used to improve the power plant start-up procedure or to increase the load change rate. Furthermore, the integration of a thermal energy storage can offer the possibility of a load shift between minimum load and full load. If a power plant is operated in minimum load - usually in times with a low spot market price - the

storage can be charged with energy from the water-steam cycle. Thus, charging the storage results in a reduction of the electrical minimum load. In times of high spot market prices the energy from the storage can be integrated into the preheating route, leading to an additional electrical power output in full load by reducing bleed steam.

The integration of such a thermal energy storage system to the underlying power plant process is shown in figure 14. The storage is charged with energy from the cold reheat steam. During discharge-mode energy from the storage system is integrated between low pressure preheater 4 and the feed water tank.

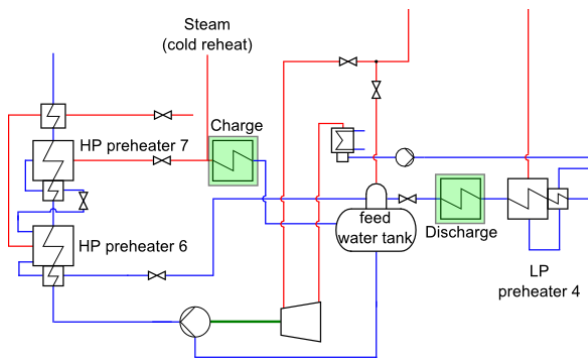


Figure 14. Extract of the process flow diagram with the integration of a thermal energy storage system

The integration of the thermal energy storage system has been realized by adding a heat consumer (charging) and a heat source (discharging) to the existing dynamic simulation model. To ensure reasonable results, a terminal temperature difference of 30 K was assumed. At this stage, the simulation stays conceptually and technologically open regarding design and used storage material. Figure 15 shows the results of the dynamic simulation model concerning the load shift between minimum load and full load as described above.

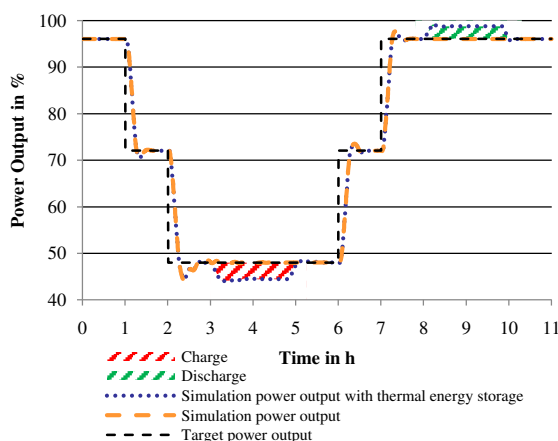


Figure 15. Simulation results for the load shift between minimum load and full load

Charging the storage system with a heat flow of $80 \text{ MW}_{\text{th}}$ leads to a reduction of the electrical minimum load in the amount of 3.7 percent. During discharge-mode the integration of $80 \text{ MW}_{\text{th}}$ enables an

additional electrical power output in full load in the amount of 2.8 percent. This corresponds to a thermal efficiency of the storage system of about 75 percent. The maximum heat flow is limited to $80 \text{ MW}_{\text{th}}$ to ensure the minimum mass flow rate of heating steam in the direction of the feed water tank (deaerator).

Based on these first results, the storage concept has to be designed in more detail in the next step. Furthermore, simulations about the increase of the load change rate by reducing the power plants dead and balancing time through the thermal energy storage system will be performed using the dynamic simulation model. Additionally, further concepts with the target of a multiple purpose storage will be developed. Multiple purpose storage means, that the concept is able to combine different flexibilization options (e.g. load change rate, primary control energy and start-up time/costs).

6.2 Load Change Rate

In this section, a simulation study is described showing a comparison of different maximum permissible rates of change \dot{P}_{perm} within the unit control and the evaluation of the resulting load change rates of the power plant process.

Figure 16 describes the determination of the load change rate with the 90 percent method [12].

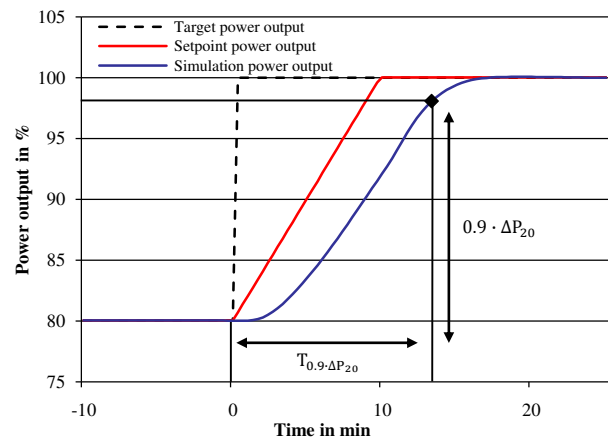


Figure 16: Determination of the load change rate with the 90 percent method

The calculation of the load change rate with the 90 percent method is based on the following equation:

$$\frac{dP}{dt} = \frac{0.9 \cdot \Delta P_{LC}}{T_{0.9 \cdot \Delta P_{LC}}} \quad (1)$$

where LC is the target load change in MW_{el} .

For the illustrated example shown in figure 16, the load change rate is calculated to

$$\frac{dP}{dt} = \frac{0.9 \cdot 20 \%}{13 \text{ min}} \approx 1.4 \%/\text{min} \quad (2)$$

That is 70 % related to the maximum permissible rate of change, represented by the red line in figure 16 ($\dot{P}_{\text{perm}} = 2 \%/\text{min} = 100 \%$).

Figure 17 shows the simulation results for a load change from 80 percent to 100 percent. The figure focuses on the period of 40 minutes, 10 minutes before and 30 minutes after the load change. The slow behavior at the beginning of the load change can be justified by the sluggish behavior of the process, in particular by the temporal behavior of the coal mills, the transport processes, the heat delivery and the heat transfer in the heating surface tubes. Due to the simplified implementation of the unit control, as explained in section 4.3, the simulated power output overshoots the target power output. Hence, the exact time curves have to be treated with caution. But the comparison of the configurations with three different maximum permissible rates of change \dot{P}_{perm} is admissible.

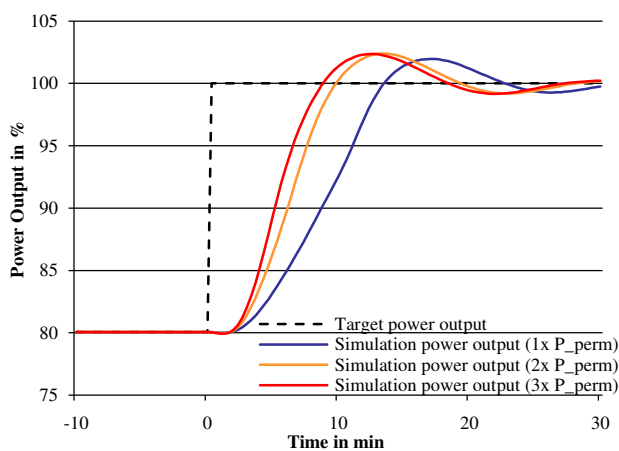


Figure 17. Comparison of the simulated power output with different maximum permissible rates of change \dot{P}_{perm} . The evaluation of the three different rates of change and the resulting load change rates (determined with the 90 % method) is presented in figure 18 for different load changes.

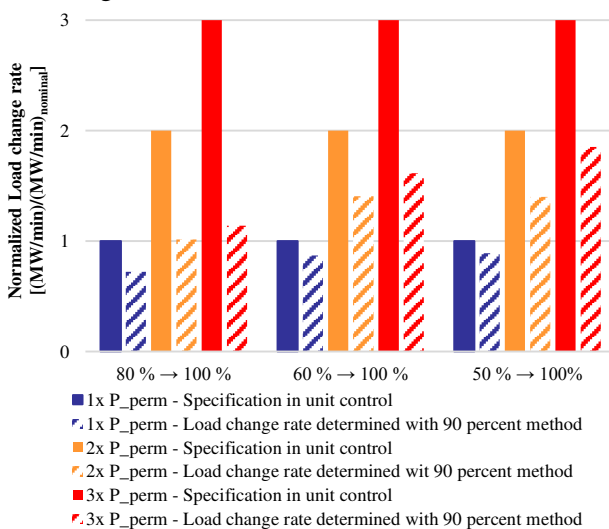


Figure 18. Evaluation of the resulting load change rates by specifying different rates of change \dot{P}_{perm}

It can be seen, that a doubling of \dot{P}_{perm} in the unit control does not lead to a doubling of the achieved load change rate of the power plant process. This fact can be explained through the slow behavior at the beginning of the load changes. Furthermore, figure 18 shows that the load change rate is higher for higher load changes, as the influence of the inertia of the power plant process is getting smaller.

7 Summary and Outlook

Through the increasing share of fluctuating renewable energies in power production the operating flexibility of conventional steam power plants becomes increasingly important. Thereby dynamic simulation models are gaining in relevance as they offer a tool to evaluate measures to face the increasing flexibility requirements.

This paper presented the development of a dynamic simulation model for a coal-fired steam power plant with the open source library *Clara* with a focus on the detailed non-steady-state modeling of the steam generator. The results of the dynamic simulation model regarding the power output as well as the mass flow rates show a good accordance to operating data during a load profile. The validation of the detailed steam generator model was also proven successful by the comparison of water-steam temperatures during the load profile.

Furthermore, first results of the integration of a thermal energy storage system for a load shift between minimum load and full load were presented. The integration of the thermal energy storage system can lead to a reduction of the electrical minimum load during charging-mode and can supply additional electrical energy during discharge-mode in full load. In addition, first results regarding the variation of the maximum permissible rate of change \dot{P}_{perm} were presented.

In the next steps, the dynamic simulation model will be developed further, especially regarding a more detailed implementation of the unit control and the extension of the thermal energy storage evaluations towards multiple purpose storages. Also further simulations about possible flexibilization measures regarding the improvement of the start-up procedure, the increase of the load change rate or the supply of (primary) control power will be performed with the dynamic simulation model.

Acknowledgements

The authors would like to thank the German Federal Ministry for Economic Affairs and Energy (BMWi) and E.ON, RWE and Vattenfall for the financial support of the project in the frame of the COORETEC program (Grant No. 03ET7017G) [13].

Gefördert durch:



aufgrund eines Beschlusses des Deutschen Bundestages

References

- [1] AG Energiebilanzen. *Bruttostromerzeugung in Deutschland von 1990 bis 2013*, 12.12.2013.
- [2] J. Brunnemann, F. Gottelt, K. Wellner, A. Renz, A. Thüring, V. Röder, C. Hasenbein, C. Schulze, G. Schmitz, J. Eiden. *Status of ClaRaCCS: Modelling and Simulation of Coal-Fired Power Plants with CO₂ capture*, 9th Modelica Conference, Munich, Germany, 2012.
- [3] ClaRa <http://www.claralib.com>
- [4] B. Epple, R. Leithner. *Simulation von Kraftwerken und Feuerungen*, Springer, Wien, 2012.
- [5] German Federal Ministry for Economic Affairs and Energy (BMWi). *Energiedaten*, Berlin, 21.10.2014.
- [6] German Renewable Energy Sources Act, current version from 01.08.2014.
- [7] German Transmission System Operators. *"Netzentwicklungsplan 2014 (erster Entwurf)"*, <http://www.netzentwicklungsplan.de/netzentwicklungsplan-2014-zweiter-entwurf>
- [8] Market simulation data from the Institute of Energy Economics at the University of Cologne in the frame of the Project Partner Steam Power Plant [13].
- [9] Mitsubishi Hitachi Power Systems Europe GmbH, *reference list*.
- [10] M. Richter, G. Oeljeklaus, K. Görner. *Dynamic Simulation of Coal-Fired Power Plants focusing on the Modeling of the Steam Generator*, 10th European Conference on Industrial Furnaces and Boilers, Porto, Portugal, 2015.
- [11] VDI/VDE 3503. *Steam Temperature Control in Fossil Fired Steam Power Stations*.
- [12] VDI/VDE 3508. *Unit control of thermal power stations*
- [13] VGB PowerTech e.V., *VGB Research Project 375: Partner steam power plant for the regenerative power generation*.
http://www.vgb.org/en/research_project375.html

Where `impact` got Going

Michael Tiller¹ Dietmar Winkler²

¹Xogeny Inc., USA, michael.tiller@xogeny.com

²Telemark University College, Norway, dietmar.winkler@hit.no

Abstract

This paper discusses the `impact` package manager. The primary goal of this project is to support the development of a healthy eco-system around Modelica. For many other languages, the existence of an easy to use package manager has made it easier for people to explore and adopt those languages. We seek to bring that same kind of capability to the Modelica community by incorporating useful features from other package managers like `bower`, `npm`, *etc.*

This paper is an update on the status of the `impact` package manager which was discussed previously in (Tiller and Winkler 2014). This latest version of `impact` involves a complete rewrite that incorporates a more advanced dependency resolution algorithm. That dependency resolution will be discussed in depth along with many of the subtle issues that arose during the development of this latest version of `impact`. Along with a superior dependency resolution scheme, the new version of `impact` is much easier to install and use. Furthermore, it includes many useful new features as well.

Keywords: `Modelica`, *package management*, *GitHub*, *dependency resolution*, *golang*

1 Introduction

1.1 Motivation

The motivation behind the `impact` project is to support two critical aspects of library development. The first is to make it very easy for library developers to *publish* their work. The second is, at the same time, to make it easy for library consumers to both *find* and *install* published libraries.

We also feel it is important to reinforce best practices with respect to model development. For this reason, we have made version control an integral part of our solution. Rather than putting users in a position to have to figure out how to make `impact` work with a version control system, we've build `impact` around the version control system. Not only do users not have to find a way to make these technologies work together, `impact` actually nudges those not using version control toward

solutions that incorporate version control. In this way, we hope to demonstrate to people the advantages of both `impact` and version control and establish both as “best practices” for model development.

By creating a tool that makes it easy to both publish and install libraries, we feel we are creating a critical piece of the foundation necessary to establish a **healthy ecosystem** for model development.

1.2 History

Earlier, we mentioned that `impact` has been completely rewritten. In fact, the very first version of `impact` was just a single Python script for indexing and installing Modelica code (Tiller 2013). It eventually evolved into a multi-file package that could be installed using the Python package management tools.

2 Requirements

After building the original Python version, we gave some thought to what worked well and what didn't work well. One issue we ran into almost immediately was the complexity of installing the Python version of `impact`. Python is unusual in that it has two package managers, `easy_install` and `pip`. It comes with `easy_install`, but `pip` is the more capable package manager. So in order for someone to install `impact`, they first needed to install Python, then install `pip` and then install `impact`. This was far too complicated. So we wanted to come up with a way for people to install `impact` **as a simple executable** without any run-time or prerequisites.

Another issue we ran into with the Python version was the fact that there are two different and incompatible versions of Python being used today (*i.e.*, 2.x and 3.x). Trying to support both was an unnecessarily inefficient use of resources. We also had some difficulties in the Python version with support for SSL under Windows (StackOverflow 2010). Because we were doing lots of “crawling” (more on this shortly), we needed a platform that provided **solid HTTP client support**. For these reasons, we felt we needed to move away from Python altogether.

Although most Modelica users run their development tools and simulations under Windows, there are several tools that support OSX and Linux as well as Windows. So as to not neglect users of those tools and to support more cross-platform options, we also wanted to be able to **compile impact for all three major platforms**.

Furthermore, we wanted to provide a simple executable for all platforms without having to have actual development machines for each of these different platforms. For this reason, **cross compilation** between different platforms was an important consideration as well.

Of course, we also wanted to have **good performance**. For most package management related functions, the speed of the internet connection is probably the biggest limiting factor. So CPU performance wasn't that high on the list. But, as we shall discuss shortly, the computational complexity of the dependency resolution algorithm we implemented could lead to some computationally intensive calculations for complex systems of dependencies.

For these reasons, we ultimately rewrote `impact` in Go (Go-Developers 2014). Go is a relatively new language from Google that stresses simplicity in language semantics but, at the same time, provides a fairly complete standard library. You can think of Go as being quite similar to C with support for extremely simple object-oriented functionality, automatic garbage collection and language level support for CSP-based concurrency. With Go, we were able to satisfy all the requirements above.

3 Version Numbering

Before we dive into all the details associated with crawling, indexing, resolving and installing, it is useful to take a moment to briefly discuss versioning. Modelica supports the notion of versions through the use of the `version` and `uses` annotations. These two annotations allow libraries to explicitly state what version they are and what versions of other libraries they use, respectively.

But there is one complication to the way Modelica deals with versions. In Modelica, a version is simply a string. This by itself isn't a problem. But it becomes a problem, as we will discuss in greater detail shortly, when you need to understand relationships between versions. In particular, there are two important things we would like to determine when dealing with version numbers. The first is an unambiguous ordering of versions. In other words, which, of any two versions, is the "latest" version? The second is whether a newer version of a library is "backwards compatible" with a previous version. These are essential questions when trying to resolve dependencies and the current string based approach to versions in Modelica is not semantically rich enough to help us answer either of these.

This issue is not unique to the Modelica world. These

same questions have been asked for a very long time and various approaches have been invented to deal with answering these questions. One recent and widely used approach is to employ what is called **semantic versioning** (Preston-Werner 2014). Semantic versioning is pretty much what it sounds like, an approach to defining version numbers where the version numbers have very explicit meanings associated with them.

A very simple summary of semantic versioning would be that **all** versions have exactly three numerical components, a major version number, a minor version number and a patch. A semantic version must have all of these numbers and they must be `.`-separated. For this reason, the following versions are not legal semantic version numbers: `1`, `1a`, `1.0`, `1.0-beta5`, `4.0.2.4096`. Each of the three numbers in a semantic version means something. If you make a non-backward compatible change, you must increment the major version. If you make a backward compatible version, you must increment the minor version. If you make a change that should be completely compatible with the previous version (*e.g.*, doesn't add any new capability), you increment only the patch version.

There are additional provisions in semantic versioning to handle pre-release versions as well as build annotations. We will not discuss those semantics here, but they are incorporated into our implementation's treatment of version numbers.

Our use of semantic versioning is aligned with our goal of strongly encouraging best practices. It is important to point out that the use of semantic versions is completely legal in Modelica. In other words, Modelica allows a wider range of interpretations of version numbers. By using semantic versions, we narrow these interpretations but we feel that this narrowing is much better for the developer since it also provides meaning to the version numbers assigned to a library.

However, because Modelica libraries are free to use nearly any string as a version number, we need to find a way to "bridge the gap" between past usage and the usage we are encouraging moving forward. Although internally `impact` understands **only** semantic versions, it is still able to work with nearly all existing Modelica libraries. This is achieved through a process of "normalizing" existing versions. When `impact` comes across versions that are not legal semantic versions, it attempts to create an equivalent semantic version representation. For example, a library with a version string of `1.0` would be represented by the semantic version `1.0.0`.

For this normalization to work, it is important to make sure that the normalization is performed *both* on the version number associated with a library and on the version numbers of the libraries used. In other words, it must be applied consistently to both the `version` and `uses` annotations.

4 Indexing

As mentioned previously, there are two main functions that `impact` performs. The first is making it easy for library developers to publish their libraries and the other is making it easy for consumers to find and install those same libraries. Where these two needs meet is the library index. The index is **built** by collecting information about published libraries. The same index is **used** by consumers searching for information about available libraries.

Building the index involves crawling through repositories and extracting information about libraries that those repositories contain. In the following section we will discuss this crawling process in detail and describe the information that is collected and published in the resulting index.

4.1 Sources

Currently, `impact` only supports crawling GitHub (GitHub 2014) repositories. It does this by using the GitHub API (GitHub-Developers 2014) to search through repositories associated with particular users and to look for Modelica libraries stored in those repositories. We will shortly discuss exactly how it identifies Modelica libraries. But before we cover those details it is first necessary to understand which *versions* of the repository it looks into.

Each change in a Git repository involves a *commit*. That commit affects the contents of one or more files in the repository. During development, there are frequent commits. To identify specific versions of the repository, a tag can be associated with that version. Each tag in the repository history that starts with a `v` and is followed by a semantic version number is analyzed by `impact`.

4.2 Repository Structure

For each version of a repository tagged with a semantic version number, `impact` inspects the contents of that version of the repository looking for Modelica libraries. There are effectively two ways that `impact` finds Modelica libraries in a repository. The first is to check for libraries in “obvious” places that conform to some common conventions. For cases where such conventions are insufficient, `impact` looks for a file named `impact.json` to explicitly provide information about the repository.

4.2.1 Conventions

With respect to `impact`, the following is a list of “obvious” places that `impact` checks for the presence of Modelica libraries:

- `./package.mo` The entire repository is treated as a Modelica package.

- `./<dirname>/package.mo` or `./<dirname> <ver>/package.mo` The directory `<dirname>` is presumed to be a Modelica package.
- `./<filename>.mo` or `./<filename> <ver>.mo` The file `./<filename>.mo` is a file containing a Modelica library.

In all cases, the name of the library is determined by parsing the actual Modelica package definition and is not related to the name of the repository. As can be seen from these conventions, only files and directories that exist at the root level are checked for Modelica content.

4.2.2 impact.json

For various reasons, library developers may not wish to conform to the repository structure patterns discussed previously. Furthermore, there may be additional information they wish to include about their libraries. For this reason, a library developer can include an `impact.json` file in the root of the repository directory that provides additional information about the contents of the repository. For example, a repository may contain two or more Modelica libraries in sub-directories. The `impact.json` file allows information about the storage location of each library in the repository to be provided by the library developer. Furthermore, the author may wish to include contact information beyond what can be extracted from information about the repository and its owner. These are just a few use cases for why an `impact.json` file might be useful for library developers. A complete schema for the `impact.json` file can be found later in Section 4.4.2.

4.3 Handling Forks

The Modelica specification implicitly assumes that each library is uniquely identified by its name. This name is used in both the `version` and `uses` annotations as well as any references in Modelica code (e.g., `Modelica` in `Modelica.SIunits`). This assumption works well when discussing libraries currently loaded into a given tool. But when you expand the scope of your “namespace” to include all libraries available from multiple sources, the chance for overlap becomes possible and must be dealt with.

Previously, we mentioned the importance of supporting best practices in model development and the specific need to accommodate version control as part of that process. Up until now, we have leveraged version control to make the process of indexing and collection libraries easier. However, version control does introduce one complexity as well. That complexity is how to deal with *forks*.

Forks are common in open source projects and typically occur when there are multiple perspectives on how

development should progress on a given project. In some cases, rather than reconciling these different perspectives, developers decide to proceed in different directions. When this happens, the project becomes “forked” and there are then (at least) two **different** libraries being developed in parallel. Each of these libraries may share a common name and perhaps even the same version numbers but still be fundamentally different libraries.

A fork can arise for another, more positive, reason. When someone improves a library they may not have permission to simply fold their improvement back into the original library. On GitHub in particular, it is extremely common for a library to be forked simply to enable a third-party to make an improvement. The author of the improvement then sends what is called a *pull request* to the library author asking them to incorporate the improvement. In such a workflow, the fork is simply a temporary measure (akin to a branch) to support concurrent development. Once the pull request is accepted, the fork can be removed entirely.

Regardless of why the fork occurs, it is important that `impact` accommodates cases where forking occurs. This is because forking is a very common occurrence in a healthy eco-system. It indicates progress and interest and we should not do anything to stifle either of these. The issue with forking is that the same name might be used by multiple libraries. In such cases, we need a better way to uniquely identify libraries.

For this reason, `impact` records **not only the library name, but also a URI associated with each library**. In this way, the URI serves as a completely unambiguous way of identifying different libraries. While two forks may have the same name, they will never have the same URI.

4.4 Schema

We’ve mentioned the kinds of information `impact` collects while indexing as well as the kind of information that might be provided by library developers (via `impact.json` files). In this section, we will provide a complete description of information used by `impact`.

4.4.1 `impact_index.json`

As part of the indexing process, `impact` produces an index file named `impact_index.json`. This is a JSON encoded representation of all the libraries found during indexing. The root of an `impact_index.json` file contains only two elements:

version A string indicating what version of `impact` generated the index. The string is, of course, a semantic version.

libraries The `libraries` field is an array. Each element in the array describes a library that was found. **The order of the elements is significant.** Libraries

that occur earlier in the list take precedence over libraries that appear later. This is important in cases where libraries have the same name.

For each library in the `libraries` array, the following information may be present:

name The name of the library (as used in Modelica)
description A textual description of the library
stars A way of “rating” libraries. In the case of GitHub, this is the number of times the repository has been starred. But for other types of sources, other metrics can be used.
uri A URI to uniquely identify the given library (when it shares a common `name` with another library)
owner_uri A URI to uniquely identify the owner of the library
email The email address of the owner/maintainer of the library
homepage The URL for the library’s homepage
repository The URI for the library’s source code repository
format The format of the library’s source code repository (e.g., Git, Mercurial, Subversion)
versions This is an object that maps a semantic version (in the form of a string) to details associated with that specific version

The details associated with each version are as follows:

version A string representation of the semantic version (*i.e.*, one that is identical to the key).
tarball_url A URL that points to an archive of the repository in `tar` format.
zipball_url A URL that points to an archive of the repository in `zip` format.
path The location of the library within the repository.
isfile Whether the Modelica library is stored as a file (`true`) or as a directory (`false`)
sha This is a hash associated with this particular version. This is currently recorded by `impact` during indexing but not used. Such a hash could be useful for caching repository information locally.
dependencies This is an array listing the dependencies that this particular version has on other Modelica libraries. Each element in this array is an object with one field, `name`, giving the name of the required library and another field, `version`, which is the **semantic version** of that library represented as a string (see previous discussion on normalization in 3).

4.4.2 `impact.json`

As mentioned previously in Section 4.2.2, each directory can include a file named `impact.json` that provides explicit information about Modelica libraries contained

in that repository. The root of the `impact.json` file contains the following information:

- owner_uri** A link to information about the libraries owner
- email** The email address of the owner or maintainer
- alias** An object that whose keys are the names of libraries and whose associated values are the unique URIs of those libraries. This information can, therefore, be used to disambiguate between dependencies where there may be multiple libraries with that name.
- libraries** This is an array where each element is an object that contains information about a library present in the repository.

For each library listed in the `libraries` field, the following information may be provided:

- name** The name of the library
- path** The path to the library
- isfile** Whether the entity pointed to by `path` is a Modelica library stored as a file (`true`) or as a directory (`false`).
- issues_url** A link pointing to the issue tracker for this library
- dependencies** An explicit list of dependencies for this library (if not provided, the list will be based on the `uses` annotations found in the package definition).

Each dependency in the list should be an object that provides the following information:

- name** Name of the required library
- uri** Unique URI of the required library
- version** **Semantic version** number of the required library (represented as a string)

5 Installation

The previous section focused on how `impact` collects information about available libraries. The main application for this information is to support installation of those libraries. In this section, we'll discuss the installation side of using `impact`.

5.1 Dependency Resolution

5.1.1 Background

To understand the abstract problem behind the concept of a dependency, we refer to the formal study undertaken in (Boender 2011). There, a repository is defined as a triple (R, D, C) of a set of packages R , a dependency function $D : R \rightarrow \mathcal{P}(\mathcal{P}(R))$, and a *conflict relation* $C \subseteq R \times R$.

At that level, version numbers have been abstracted to (distinguishable) packages: Every version yields a distinctive package $p \in P$.

The dependency function D maps a package p to sets of sets of packages $d \in D(p)$, where each set represents a way to provide one required feature of p . In other words: If for each $d \in D(p)$ at least one package in d is installed, it is possible to use p .

Currently, there is no way to express *conflicts* directly in a Modelica package. However, due to the existence of external libraries (which could conflict in arbitrary ways), it is likely that such a need will arise in the future. Additionally, current Modelica makes it impossible to refer to two different versions of a library from the same model. Hence, we consider different versions of the same package conflicting.

The dependency resolution of `impact` fits into Boender's model. Therefore, the conclusions drawn in (Boender 2011) can be applied to `impact` as well:

The set of packages `impact` installs for a given project needs to fulfill two properties, Boender calls *abundance* and *peace*. Informally, abundance captures the requirement that all dependencies be met while peace avoids packages that are in conflict with each other. A set of packages that is peaceful and abundant is called *healthy* and a package p is called *installable* w.r.t. a given repository if and only if there exists a healthy set I in said repository such that $p \in I$.

The problem of finding such an installable set is however a hard one. In fact, Boender proves by a simple isomorphism between the boolean satisfiability problem and the dependency resolution that finding such a set is NP-hard. Fortunately, for the current typical problem size, this isn't really an issue.

5.1.2 Resolution Algorithm

The indexing process collects quite a bit of information about available libraries. Most of the complexity in implementing the installation functionality in `impact` is in figuring out **what** to install. And most of that complexity is in finding a set of versions for the required libraries that satisfy all the dependency relations. This process is called dependency resolution.

The resolution algorithm starts with a list of libraries that the user wants to install. In some cases, this may be a single library but, in general, the list can be of any length. For each library in the list, the user may specify a particular version of the library they wish to install, but this isn't mandatory. One important point here is that we refer to this as a *list*, not a set. Order is significant here. The libraries that appear first are given a higher priority than those that appear later.

Let's explain why this priority is important. Consider a user who wishes to install libraries A and B. If the user has not explicitly specified what version of each library they are interested in, `impact` assumes the user wants

the latest version, if possible. But what if the latest version of *both* cannot be used? To understand this case, consider the following constraints:

A:1.0.0 uses B:2.0.0

A:2.0.0 uses B:1.0.0

where A:1.0.0 means version 1.0.0 of library A. This example is admittedly contrived, but the underlying issue is not. We can see here that if we want the latest version of A, we cannot also use the latest version of B (and *vice versa*) while still honoring the constraints above. The ordering of the libraries determines how we “break the tie” here. Since A appears first, we assume it is more important to have the latest version of A than to have the latest version of B.

Let’s take this extremely simple example to outline how the resolution algorithm would function in this case. In later sections, we’ll introduce additional complexities that must be dealt with.

If a user asks for libraries A and B to be installed, the question that the dependency algorithm has to answer is **which versions** do we use. Assuming that each library has a version 1.0.0 and 2.0.0, then each “variable” in this problem has two possible values. The following table essentially summarizes the possibilities:

Version of A	Version of B
1.0.0	1.0.0
1.0.0	2.0.0
2.0.0	1.0.0
2.0.0	2.0.0

This is a simple enumeration of the possibilities. But *remember*, we assume the user wants the most recent version *and* we assume A is more important than B. Semantic versioning provides us with a basis for determining which version is more recent. Given these we reorder these combinations so that the most desirable combinations appear first and the least desirable appear last:

Version of A	Version of B
2.0.0	2.0.0
2.0.0	1.0.0
1.0.0	2.0.0
1.0.0	1.0.0

Now we see the impact of the dependency constraints. Specifically, the first (most desirable) combination in this table does not satisfy the dependency constraints (*i.e.*, A:2.0.0 does not work with B:2.0.0). If we eliminate rows that violate our dependency constraints, we are left with:

Version of A	Version of B
2.0.0	1.0.0
1.0.0	2.0.0

In summary, we order the combinations by their desirability (considering both the relative priority of the

libraries and their version numbers) and then we eliminate combinations that don’t satisfy our dependency constraints.

This gives an overview of how the algorithm works conceptually. But, as you may have guessed, the problem is not quite this simple. Consider now a slightly more complex case with the following dependencies:

1	A:3.0.0	uses	B:1.2.0
2	A:3.0.0	uses	C:1.1.0
3	B:1.2.0	uses	C:1.2.0
4	A:2.0.0	uses	B:1.1.0
5	A:2.0.0	uses	C:1.0.0
6	B:1.1.0	uses	C:1.1.0
7	A:1.0.0	uses	B:1.0.0
8	A:1.0.0	uses	C:1.0.0
9	B:1.0.0	uses	C:1.0.0

Now we have three variables we need to solve for, A, B and C. For each variable, we have three possible values. As we’ve already described, newer versions are preferred over older versions while searching. This means that the first combination we will consider will be...

A:3.0.0, B:1.2.0 and C:1.2.0

...and the last combination we will consider will be...

A:1.0.0, B:1.2.0 and C:1.2.0

There are several interesting things to notice about this case. First, although the problem is not particularly large (3 libraries with 3 versions each), the number of combinations to check is significant (*i.e.*, $3 \cdot 3 \cdot 3 = 27$). Of these 27 combinations, only the last one to be considered (*i.e.*, the least desirable) satisfies the dependency constraints. There is nothing we can really do about the fact that the oldest version of each of these libraries must be used (this is dictated by the dependencies themselves and has nothing to do with the algorithm). But the complication is that we must consider all of them (in this contrived case) before finding the one we want.

In reality, we would not actually enumerate all possibilities *a priori*. Instead, we would simply consider each “variable” one at a time and loop over all possible versions. If, at any point, we find a conflict with our constraints, we simply break out of the inner most loop. This is referred to as *backtracking*. In Modelica pseudo-code, the algorithm (for this specific case) might look like this:

```

for A in ["3.0.0", "2.0.0", "1.0.0"] loop
  for B in ["1.2.0", "1.1.0", "1.0.0"] loop
    if not are_compatible(A,B) then
      break;
    end if;
  for C in ["1.2.0", "1.1.0", "1.0.0"] loop
    if not are_compatible(B,C) then
      break;
    end if;
    if not are_compatible(A,C) then
      break;
    end if;
    //If we get here, we have a solution
  end for;
end for;
end for;

```

Using this backtracking, we can more efficiently traverse the possibilities by eliminating lots of cases that we know are a dead end (especially in larger problems). Any search based on backtracking is vulnerable to poor performance under certain (typically pathological) conditions. We'll return to this point later when we talk about performance of our current implementation.

There is one last complication we must deal with when resolving dependencies. Consider the following simple set of dependencies:

- 1 A:2.0.0 uses B:1.2.0
- 2 A:2.0.0 uses C:1.1.0
- 3 B:1.2.0 uses C:1.2.0
- 4 A:1.0.0 uses B:1.1.0 or B:1.0.0
(*i.e.*, A can use B:1.1.0 or B:1.0.0)
- 5 A:1.0.0 uses D:1.1.0
- 6 B:1.0.0 uses C:1.1.0
- 7 C:1.2.0 uses D:1.0.0
- 8 B:1.1.0 uses C:1.2.0

We can also represent this set of dependencies graphically as shown in Figure 1. Graphically, we have a box to represent each library and that box contains the different versions available. These versions are connected by the constraints shown in the table above.

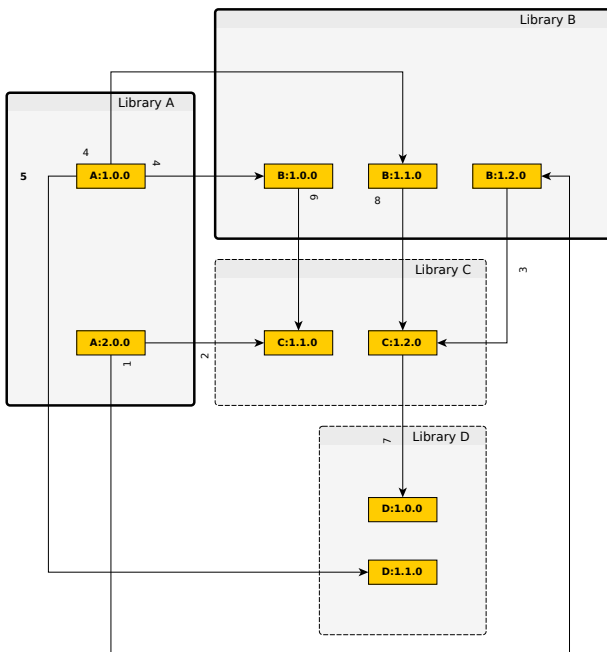


Figure 1. Graphical representation of package dependencies

Given these dependencies and the fact that the user wishes to install both A and B, what are the variables in our dependency resolution algorithm? Obviously, we must consider all the versions of both A and B (*i.e.*, we must pick a version from the box for library A and B in Figure 1). But what about C and D? It makes no sense to enumerate all combinations of versions for these four libraries because in many cases D isn't even required. Furthermore, what is their relatively priority (*i.e.*, if a choice

is required, is it more important to have the latest version of C or D?)

When resolving dependencies, we only introduce new libraries when necessary (*i.e.*, if they are needed by our current choices of existing libraries) and their relative priority is determined by the relative priority of the library that introduced them.

To understand how the resolution works in this case, first consider the case of A:2.0.0. This version cannot be chosen. This is because A:2.0.0 wants C:1.0.0 while B:1.2.0 wants C:1.1.0. So no choice for C is valid. Furthermore, we don't even consider D because it isn't required in any of these cases.

Now if we move to the case of A:1.0.0, things are more complicated. Now we **do** need to consider both D and C. However, note that because A:1.0.0 depends directly on D, we consider D more important. This is important because when considering A:1.0.0 we have two versions of B that are compatible¹ (*i.e.*, B:1.1.0 and B:1.0.0). Given that we are considering A:1.0.0 and we've already ruled out B:1.2.0, we are left with the following combinations:

Version of B	Version of D	Version of C
1.1.0	1.1.0	1.2.0
1.1.0	1.1.0	1.1.0
1.1.0	1.0.0	1.2.0
1.1.0	1.0.0	1.1.0
1.0.0	1.1.0	1.2.0
1.0.0	1.1.0	1.1.0
1.0.0	1.0.0	1.2.0
1.0.0	1.0.0	1.1.0

Notice the ordering of the columns? Since the user originally asked for both A and B, B comes first. But when it comes to C and D, having a more recent version of D is more important than having a more recent version of C.

As mentioned previously, we don't construct every combination. Furthermore, we don't always consider all libraries. The best way to understand how this search proceeds is to enumerate the partial combinations that our search generates and the point at which backtracking occurs. In such a case, we can think of the search as proceeding as follows:

¹It is not possible to express this kind of "or" dependency currently in Modelica, but it is supported by *impact*. This capability exists in *impact* both to support anticipated future capabilities in Modelica (Tiller 2012) and/or to support cases we will discuss shortly that consider cases of compatibility implicit in semantic versions.

```

A:2.0.0           I👍
A:2.0.0 & B:1.2.0 I👍
A:2.0.0 & B:1.2.0 & C:1.2.0 X→#2
A:2.0.0 & B:1.2.0 & C:1.1.0 X→#3
A:2.0.0 & B:1.1.0 X→#1
A:2.0.0 & B:1.0.0 X→#1
A:1.0.0 & B:1.2.0 X→#4
A:1.0.0 & B:1.1.0 I👍
A:1.0.0 & B:1.1.0 & D:1.1.0 I👍
A:1.0.0 & B:1.1.0 & D:1.1.0 & C:1.2.0 X→#7
A:1.0.0 & B:1.1.0 & D:1.1.0 & C:1.1.0 X→#8
A:1.0.0 & B:1.1.0 & D:1.0.0 X→#5
A:1.0.0 & B:1.0.0 & D:1.1.0 & C:1.2.0 X→#6
A:1.0.0 & B:1.0.0 & D:1.1.0 & C:1.1.0 ✓

```

This elaboration of the search shows the role that the relative priority of libraries and versions has on the search order but also how a particular library is not even considered until a dependency on that library is introduced by choosing a particular version that depends on it.

It should be noted that there are a variety of other special cases we also deal with like self dependency and cyclic dependency. But these are constraints like any other and don't really impact the algorithm in any significant way.

The actual algorithm is implemented by the `findFirst` method on the `LibraryGraph` type found in the `github.com/xogeny/impact/graph` package. The inputs to this function are:

- mapped** Any existing decisions about specific versions of each library (initially empty)
- avail** The set of all (remaining) possible versions for each library (initially all versions of all libraries)
- rest** A list of libraries that are required based on existing decisions but for which no version choice has yet been made (initially the libraries the user wants installed in the order specified by the user)

The algorithm then proceeds as follows:

1. Is `rest` empty? If so, we are done and we have a solution (*i.e.*, `mapped`)
2. Consider the first library in `rest`
3. Loop over available versions (based on `avail`)
 - (a) Add this choice to `mapped`
 - (b) Find any new library dependencies resulting from this choice
 - (c) If incompatible decisions have already been made about these new dependencies, backtrack
 - (d) Update `avail` to include version of new dependencies that are compatible with our previous choices

- (e) If there are no possible versions for any library we depend on, backtrack
- (f) Return the result of calling this function again recursively using updated values for `mapped`, `avail` and `rest`.

5.1.3 Formulating Constraints

The default assumption is that dependencies will come from the `uses` annotation in Modelica. There is a proposal to extend the `uses` annotation to allow multiple compatible versions to be listed (*vs.* only a single compatible version today). As mentioned previously, such an `or` relationship is already supported by `impact`. So this change would not impact the resolution algorithm used by `impact`.

Although it hasn't yet been implemented, one proposed fallback mode for `impact` is to ignore the explicit dependencies contained in Modelica code and instead rely on the dependency relationships **implicit** in semantic versions. In other words, if a library B has two versions, 1.1.1 and 1.1.2, and those versions strictly follow semantic versioning conventions, then we know that any library that depends on B:1.1.1 must also be compatible with B:1.1.2. Such a fallback mode could be employed when `impact` is unable to find a solution using explicit constraints.

6 Go Implementation

We've created an implementation of `impact` using Go. This implementation includes different sub-packages for dealing with crawling repositories, resolving dependencies, parsing Modelica code and managing configuration settings. It also contains a sub-package for implementing the command-line tool and all of its sub-commands. This structure means that `impact` is not only a command-line tool, but also a Go library that can be embedded in other tools.

The Go implementation includes the following commands:

- search** Search library names and descriptions for search terms.
- install** Install one or more libraries and their dependencies.
- index** Build an index of repositories.
- version** Print out version and configuration information.

For each command, you can use the `-h` switch to find out more about the command and its options.

Earlier we described our requirements. The main reason we moved to Go from Python was Go's support for cross-compiling between all major platforms and the fact

that it generates a statically linked binary that doesn't depend on any runtime. The Go compiler includes a complete implementation of HTTP for both the client and server. In fact, the standard library for Go is fairly complete. At the moment, the only third party dependencies for `impact` are a Go implementation of the GitHub v3 API and an implementation of semantic versioning.

The performance of compiled Go code is quite good. In Section 5.1.2 we described how the algorithm we are using could, in a worst case scenario, search every potential combination before finding either a solution or failing. We constructed several test cases with n variables where each variable had 2 possible values. The result is that there will be 2^n possible combinations. These cases were contrived so that the least desirable combination was the only one that would satisfy the dependency constraints. We tested the time required for find a solution for different values of n and we got the following performance results:

n	Time (ms)
10	45
12	141
14	646
20	52,000

It is important to keep in mind that this is a **contrived** case to demonstrate the worst possible case for resolution. There may very well be other algorithmic approaches that will find identical solutions but search more efficiently. But given what we know about Modelica libraries and their dependencies, we found this performance more than sufficient for our application.

One last point worth making about the implementation of `impact` has to do with security. In order to generate an index from GitHub repositories, it is necessary to crawl repositories. In order to accomplish this, many API calls are required. GitHub will only allow a very limited number of "anonymous" API calls. This limit will be reached very quickly by `impact`. In order to increase the number of allowed API calls, GitHub requires an "API key" to be used. Such a key can be provided to `impact` but it cannot be provided via a command line option or a configuration file. This is to avoid this sensitive information being inadvertently recorded or exposed (e.g., by committing it to a version control repository). Instead, such tokens must be provided as environment variables.

The `impact` source code is licensed under an MIT license and is hosted on GitHub. The GitHub repository (Xogeny 2015) includes a `LICENSE`, `README.md` and `CONTRIBUTING.md` which provide a detailed license, introductory documentation and instructions for contributors, respectively. We've linked the GitHub repository to a continuous integration service so that each commit triggers tests and emails out build status to the maintainers.

7 impact on library developers

What does all this mean for library developers wanting to make their library accessible via `impact`? Let us first have a look at the past "sins" that were restricting the development work on Modelica libraries.

7.1 Observations

1. We noticed that the `MODELICAPATH` concept is not properly understood by the users and often gets in their way. Therefore we should not rely on it but rather work with all of our files collected into a *working directory* (which should always part of the `MODELICAPATH` and made first priority for the look-up in the tool).
2. If we go away from having to collect all Modelica libraries in the `MODELICAPATH` then there is no longer a need to store the version number with the library folder name. I.e., simply "`<PackageName>`" is sufficient and no need for "`<PackageName> <Version>`".
3. Until now, we advised the lib developers to keep the current development version in a `master` branch and merge `master` into a `release` branch where the directory structure can be changed (e.g., into "`<PackageName> <Version>`" and any generated content can be added). Finally, developers should then place a tag on the release branch. This was done for the following reasons:
 - The link to the tag provided a (tar)zip file that contained the library with the "`<PackageName> <Version>`" format ready to be used with `MODELICAPATH`. However, we no longer need to rely on `MODELICAPATH` any more we don't need to add the `<Version>` identifier to the folder anymore.
 - If we would like to see what stage a certain release in `master` was at, then we needed to either inspect the `git` history (following backwards from the release tag) or use an additional tag (e.g., "`1.2.3-dev`") which is rather cumbersome and seems unnecessary.

But since GitHub also supports new alternatives (see below) there is no longer a need for a specific `release` branch. That is to say, library developers can still use it if they think it useful but they don't have to anymore.

7.2 Repository structure recommendations

There are new features/mechanisms made available both by GitHub and `impact`:

- GitHub’s support for assets (GitHub-Blog 2013) allows us to upload additional files to tagged releases
- `impact` does not use the `MODELCPATH` model but rather uses a “one working directory per project” approach where (one version) of all required libraries and their dependencies live in one (working) directory.

We recommend that library developers make the most out of the new features above and change the structure in which they organize their library repositories.

1. Get rid of the `release` branch as long as it was only for the sake of providing a download-able zip-file with a customized structure or providing additional generated files. Instead use the new GitHub Releases (GitHub-Blog 2013) which allows uploading of additional assets for download.

- E.g., rather than adding HTML documentation to the `Resources` sub-folder and committing this to the `release` branch and then tagging it, tag the `master` branch and then generate a zip-file which contains that state and add the generated files to the tagged release. GitHub also provides some information on “Creating Releases” (GitHub-Help 2015a) and there exists, for example, the `aktaiu/github-release` tool (Hillegeer 2015) to help automating that process.
- Another benefit of the release assets is that the GitHub API (GitHub-Developers 2014) allows you to get the download count for your releases (GitHub-Help 2015b). This was not possible for the simple tagged-zip-ball downloads.

2. Get rid of the `<PackageName> <Version>` formatted folder names. The version number does not belong in the `master` (i.e., development) branch anyway and the version annotation is contained in the `version` annotation which tools will happily display for you. When you install a package with `impact` it will strip that version number in any case.

7.3 Changes for the library listing

The listing of Modelica libraries on <https://modelica.org/libraries> is generated by parsing the GitHub API and creating a static HTML file that contains all information with links. Currently it is a stand-alone *Python* script but we are thinking of adding this functionality as a sub-module to `impact` itself.

Up to May 2015 the listing pointed directly to the (tar)zip-ball URL of the latest tag of a library. This worked fine if the library used the old `release` branch

model where the “ready to install” version was placed. Clicking on that coloured version link resulted in a direct file download.

This has now been changed in such a way that if one clicks on the listed “Last Release” button one will get redirected to the “Releases” page of that project showing the last release. This has the advantage that one does not immediately download the (tar)zip ball but gets to see proper release notes first *and* is given a choice of what version of a release to download (e.g., pure source distribution of that tag, customized version with additional files, different platform dependent versions with pre-compiled binaries).

7.4 Which license is best for your library

The *Modelica Standard Library* (Modelica) (Modelica Association 2013) is licensed under the “Modelica License Version 2.0” (Modelica Association 2008). So in order to stay compatible with the `Modelica` library most user libraries chose the same license. This seemed like a natural choice. However, there is one problem which is not immediately apparent to most library developers. This is that the “Modelica License Version 2.0” contains the section “4. Designation of Derivative Works and of Modified Works” which says that:

“... *This means especially that the (root-level) name of a Modelica package under this license must be changed if the package is modified ...*”.

This clause makes perfect sense for a main library like the `Modelica` library that is developed and maintained by a major group centrally and wants to protect its product name. But what does this mean for open-source projects that no longer are hosted centrally but rather decentralized on platforms like GitHub and GitLab but where contributions no longer are made by committing directly into **one** central repository? In the de-centralized case contributions are given by first “forking” (i.e., generating a copy of the original repository), modifying that fork and then sending the contribution back via a “pull-request” (i.e., offering the originating project to accept the changes made on the fork). The problem is that the very first step of “forking” the library generated a copy with the identical “(root-level) name” and at a different location. One could argue that this alone is already a violation of the terms of the “Modelica License Version 2.0”.

So what should the library developer do? The simplest solution is to **not** use the “Modelica License Version 2.0” for libraries but rather go for standard licenses (Open Source Initiative 2015) that are more compatible with open source, community driven development (e.g., MIT or BSD licenses). Interestingly, the old “Modelica License Version 1.1” is still suitable for user libraries since it does not contain the restrictions of having to change the package name.

So what about “copyleft style” licenses? The most

famous copyleft license is the GNU General Public License. People might think this would be a good choice for a license in order to protect parts of their library from being used inside proprietary libraries without any bugfixes and improvements being fed back to them as “upstream” developers. Unfortunately the GPL also forbids that any other non-GPL library (even the *Modelica Standard Library*) uses the GPL licensed library and is distributed that way. So what about the LGPL, this allows the usage and distribution *alongside* with other non-gpl libraries. The problem here is that it does not allow static linking. Something that typically happens when one creates a compiled version of a simulation model that uses different Modelica libraries. A typical example would be the generation of an FMU (Modelica Association 2015). A way out of this is the “Mozilla Public License” which is very much alike the LGPL but allows generated code to be statically linked together with non-GPL licensed code.

In conclusion, libraries should, if possible, avoid the “Modelica License Version 2.0” as this was primarily designed for the requirements of the *Modelica Standard Library*. Perhaps there will be a future revision that is adapted to current open-source development models. But until then, we suggest the use of standard licenses along the lines of BSD/MIT or MPL.

8 Future Development

8.1 Dependency Constraints

As already mentioned, there is currently no way to express conflicts between different packages. However, it is highly likely that such conflicting pairs will exist as more and more packages are published. For instance, two Modelica models might depend on different, specific versions of an external library that cannot be linked or loaded at the same time, an already published package might contain known bugs etc. Hence, *impact* could be extended by the means to express conflicts as well.

Boender introduces the notions of *strong dependencies* and *strong conflicts* to optimize the handling of very large repositories. This kind of optimization might not be necessary in the Modelica ecosystem right now, but could provide helpful performance enhancements in future versions of *impact*.

8.2 Crawling

At the moment, *impact* is only able to crawl GitHub repositories. There is nothing particularly special about GitHub and/or its APIs. The authors are confident that indices could be constructed for many different storage types. The most obvious next steps for crawling support would be to add support for GitLab and Bitbucket (Mercurial and Subversion) repositories. Pull requests to introduce such functionality are welcome.

On a related note, we anticipate there will be many use cases where *impact* could be useful for closed source projects that involve private repositories. We think this is an important use case and we hope to provide support for crawling such repositories. This would, for example, allow model developers at companies that have made a significant investment in building Modelica related models and libraries to use *impact* to search and install these proprietary libraries via their corporate intranet.

8.3 Project Details

We have already created a number of issues that require users to provide more explicit information about how they want *impact* to function on a per project basis. For example, when working with forked libraries (where the index contains multiple libraries with the same name), it is useful to use the URIs associated with each library in the index to disambiguate which particular library to use. Furthermore, there may be cases where the user is actually interested in doing development work on the dependencies as well. In such cases, those dependencies shouldn’t simply be installed, they should be **checked out** from their repository to make modifying and re-committing easier.

For these and other project related features, we feel there is a need to introduce another file to provide such additional information that is project specific.

8.4 Web Based Search

Other package managers often provide a web site where users can search for a specific package through the web, read documentation, log issues and/or even download the packages. Because *impact* is organized into libraries (and not just a command line tool), we feel this kind of functionality could be added in the future.

8.5 Installers

Finally, when installing software, it is common for developers to distribute “installers” (*i.e.*, executables that, when run, unpack and install the software). Another potential extension of *impact* could be to generate such installers. In this case, we could once again leverage Go’s static executable generation to build such installers from the index. Instead of installing the needed files locally, the installer could simply bundle them up and attach them to an installation program using one of the many Go extensions (Riemer 2015; Tebeka 2015) for concatenating static content onto executables or simply downloading some pre-specified libraries over the network.

9 Conclusion

In conclusion, `impact` leverages information already available in Modelica source code along with some common conventions in order to help users find and install Modelica libraries. It does this by crawling repositories and indexing their contents. An index of publicly available libraries created by `impact` is hosted on `modelica.org` for use by the `impact` command line tool.

If present, the `impact` command line tool is already used by OpenModelica to help find and install dependencies. By making the `impact` executables available across platforms and providing a version of the source code that can also be embedded as a library, we hope the Modelica community will benefit from having first class package management capabilities, just like other software eco-systems.

10 Acknowledgements

The authors would like to thank Christoph Höger of Technische Universität Berlin, Martin Sjölund of Linköping University, Francesco Casella of Politecnico di Milano and Peter Harman of ESi Group for their contributions to this project.

References

- Boender, Jaap (2011). “A formal study of Free Software distributions”. PhD thesis. Université Paris-Diderot-Paris VII.
- Go-Developers (2014). *The Go Programming Language Specification*. URL: <http://golang.org/ref/spec>.
- GitHub (2014). *Build software better, together*. URL: <https://github.com/>.
- GitHub-Blog (2013). *Release Your Software*. URL: <https://github.com/blog/1547-release-your-software>.
- GitHub-Developers (2014). *GitHub API v3*. URL: <http://developer.github.com/v3/>.
- GitHub-Help (2015a). *Creating Releases*. URL: <https://help.github.com/articles/creating-releases/>.
- (2015b). *Getting the download count for your releases*. URL: <https://help.github.com/articles/getting-the-download-count-for-your-releases>.
- Hillegeer, Nicolas (2015). *aktaugithub-release*. URL: <https://github.com/aktaugithub-release>.
- Modelica Association (2008). *Modelica Licence Version 2.0*. URL: <https://modelica.org/licenses/ModelicaLicense2>.
- (2013). *Modelica - Free library from the Modelica Association*. URL: <https://github.com/modelica/Modelica>.
- (2015). *Functional Mock-up Interface*. URL: <https://fmi-standard.org>.
- Open Source Initiative (2015). *Licenses*. URL: <http://opensource.org/licenses/>.
- Preston-Werner, Tom (2014). *Semantic Versioning 2.0.0*. URL: <http://semver.org/>.
- Riemer, Geert-Johan (2015). *go.rice*. URL: <https://github.com/GeertJohan/go.rice>.
- StackOverflow (2010). *How to install Python ssl module on Windows?* URL: <http://stackoverflow.com/questions/2261866/how-to-install-python-ssl-module-on-windows>.
- Tebeka, Miki (2015). *nrsc - Resource Compiler for Go*. URL: <https://bitbucket.org/tebeka/nrsc>.
- Tiller, Michael (2012). *Modelica Change Proposal For Package Handling*. URL: https://trac.modelica.org/Modelica/raw-attachment/ticket/573/Package-Proposal_asMCP.doc.
- (2013). *Gist of first version of impact.py*. URL: <https://gist.github.com/xogeny/fac3ea9174e74275e7fe>.
- Tiller, Michael and Dietmar Winkler (2014). “`impact` - A Modelica Package Manager”. In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Ed. by Hubertus Tummescheit and Karl-Erik Årzén. Modelica Association. Linköping University Electronic Press, pp. 543–548. URL: <http://www.ep.liu.se/ecp/096/057/ecp14096057.pdf>.
- Xogeny (2015). *impact code repository on GitHub*. URL: <https://github.com/xogeny/impact>.

Visualizing Simulation Results from Modelica Fluid Models Using Graph Drawing in Python

Marcus Fuchs Rita Streblov Dirk Müller

RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Aachen, Germany, mfuchs@eonerc.rwth-aachen.de

Abstract

Models of large thermo-fluid networks can be useful to better understand the dynamic behavior of complex systems. Yet, numerical outputs and line plots of individual variables may not be sufficient ways of processing the simulation results for the user. Thus, the aim of this paper is to present a visualization approach by means of graph drawing. To demonstrate the approach, we use an example from the Modelica Standard Library and the use case of a district heating system model. We parse the Modelica model code to generate a `System` graph that represents the model structure and its graphical layout. The graph drawing subsequently visualizes the results for every time-step. In the examples, we vary line thickness to visualize mass flow rates between two nodes and line color to show temperatures of the medium. We argue, that this approach can be a useful tool for modeling and analysis.

Keywords: Visualization, Graph Drawing, Modelica Fluid, District Energy System

1 Introduction

One reason for using the Modelica modeling language is the high re-usability of component models from model libraries. In this context, the acausal connections between component models can be used to efficiently assemble larger system models (Dizqah et al., 2015). For thermo-fluid systems, the `Modelica.Fluid` (Casella et al., 2006) package includes the concept of stream connectors, which facilitates the modeling of flow networks with possible flow-reversals. In energy systems modeling, e.g. for building or district heating systems, this enables the assembly of large system models from only a limited number of component models like pumps and pipes.

When connecting multiple `Modelica.Fluid` component models in a pipe network, the fluid flow is driven by pressure differences between connectors. Often, models provide a relationship between mass flow rate and the pressure drop between the component's ports. This leads

to a network of mass flows between different pressure levels. In many cases, another key aspect of modeling are the thermal properties of the fluid flow and parts of the components. A system model containing information about all these aspects can be very useful to understand the system's dynamic behavior. Yet, with increasing system size this amount of data increases at a rate that can make it hard to comprehend and verify simulation results. In these cases, numerical outputs and line plots of individual variables may not be sufficient ways of processing simulation results for the user. Thus, the aim of this paper is to present an approach to visualize the information from thermo-fluid system simulations by means of graph drawing and animation.

The need for additional visualization approaches when dealing with complex Modelica system models and its advantages for the user's understanding has been highlighted before. Previous work on this topic has mainly focused on 3D visualization. To this end, Höger et al. (2012) present an approach called `Modelica3D`, in which Modelica code is used to communicate with 3D rendering tools. They show the applicability of this approach for multi-body systems as well as in a building energy system context, with a focus on the 3D visualization of each component. In addition, Hellerer et al. (2014) give a wide range of examples for their `DLR Visualization Library` with a focus on multi-body simulations. Both these papers also give a similar overview of other previous work on this topic. Furthermore, simulation environments like `Dymola` offer functionalities for plotting and animations of 3D objects, also with a focus on multi-body animations.

In addition to the focus on multi-body and 3D visualization, the field of thermo-fluid modeling has also in part relied on post-processing simulation results using the programming language Python. As a result, there are several Python packages with different functionalities available. One such package is `BuildingsPy` (LBL-SRG, 2015), which among other functionalities contains methods for managing simulations, unit testing model libraries, and processing result files. The package `awesim` (De Conick, 2015) is a tool that helps to manage a variety of simulations and result files and so is use-

ful for multiple simulations and parameter studies. In addition, the package `ModelicaRes` (Davies, 2015) provides a user-friendly approach to process and plot simulation results. There are thus various approaches to read and work with Modelica simulation results in Python.

When considering the processing of simulation results for thermo-fluid networks, a promising method to represent the model structure is in graph format. In a non-Modelica context, e.g. Fang and Lahdelma (2014) use a graph notation to describe a district heating network with pipe elements as edges connecting the network nodes. One useful approach to use such graph notation in Python is to use the package `networkX` Hagberg et al. (2008). `networkX` is a free package providing data structures and algorithms for different graph types and work involving different kinds of complex networks. Furthermore, its open design allows for a wide variety of data to be represented by nodes and edges. Together with the powerful plotting package `matplotlib` (Hunter, 2007), `networkX` can be used to visualize graphs in many ways.

Building on the previous work done by the Python developers mentioned above, we set out to present an approach for visualizing the dynamic behavior of complex thermo-fluid networks modeled in Modelica by means of a Python post-processing.

2 Process Overview

The approach presented in this paper aims at producing a visual output, helping to better comprehend and analyze the data produced by simulating complex thermo-fluid networks in Modelica. To this end, we use a post-processing routine in Python. Python was chosen as a programming language, in part because of its accessibility through easy syntax, wide use, and being platform-independent. Another advantage of using Python is the possibility to build on the previous work done in post-processing Modelica results as described in section 1.

Fig. 1 shows a schematic representation of the approach presented in this paper. The information contained in a Modelica model is used to initialize a `Model` graph object. As it is often helpful to make abstractions from the original model design for visualization, this `Model` graph is transformed to a `System` graph in a subsequent step. After reading the Modelica model's simulation results to the `System` graph, this class can generate a visual output in the form of static plots and video animations. Both the python classes for the `Model` and the `System` extend the class `nx.Graph` from `networkX`, so that it inherently has all of `networkX`'s well established functionalities for graph handling and analysis. In order to read the Modelica result files and process the data, the code uses the `ModelicaRes` package. This way, the `Model` and `System` classes can be focused on performing the vi-

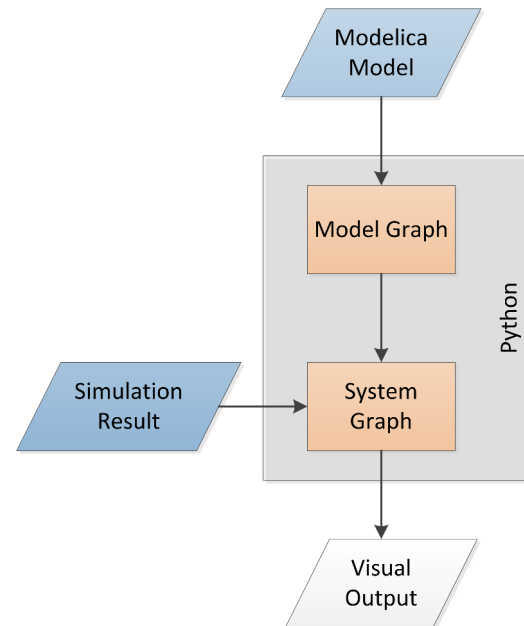


Figure 1. Flow chart for creation of visual output from Modelica model

sualization without the overhead of reproducing graph and result handling functionalities already available elsewhere.

Reading information from a Modelica model to the `Model` class in Python marks the start for the described process. This information is represented in the graph by placing edges between the nodes. In order to arrive at a more intuitive display of the model structure, especially for complex pipe networks, the `Model` graph is transformed to a `System` graph. One major change in that transformation is the introduction of network nodes between sub-models. In a further step, pipe models are transformed from individual nodes to edges connecting the network nodes. With pipes serving as connecting elements in real-world systems, this representation may be more user-friendly for the following visualization. The process is described in more detail in section 3.

As a second input to the visualization process, the `System` class uses methods from `ModelicaRes` to read data from the result file into Python. This data can be selected according to the purpose of the visualization. Yet, for analyzing thermo-fluid systems, we will concentrate on the processing of mass flow rates, pressures, enthalpies, and temperatures. In order to handle this data efficiently, `networkX` allows to attach almost any kind of data and objects to individual nodes and edges. Thus, each node and edge representing a model component can hold its relevant information from the Modelica result file. As the dynamic behavior of the system is of special interest, each dataset contains the time-series of data for every time-step of the simulation.

The data attached to nodes and edges can subsequently be used to visualize the overall system behavior

by means of graph drawing. In addition to static graph drawings showing the graph's structure, the information contained in the graph drawing can be extended by different means. For this demonstration, we will use the line thickness and color of edge connections to represent mass flow rates and temperatures for every simulation time-step. In section 7 we will point to further possibilities of enriching this data visualization approach in future work.

After visualizing the system properties for every time-step, we create a video from the individual plots, which as a final outcome produces an accessible and intuitive way to animate an amount of data for a complex system that would be hard to process for a human user in a standard 2D line plot. In the following sections, we will present the individual steps outlined above in more detail for an example model from the Modelica Standard Library. After that, we will present a use case of a campus-scale district heating network to demonstrate the capabilities of the visualization approach in an applied context.

3 Translation of Modelica Model to Graph

As outlined above, the developed `Model` class aims at a representation of the Modelica model in a graph structure using Python. We will use the `model IncompressibleFluidNetwork` from the Modelica Standard Library's `Modelica.Fluid.Examples` package to illustrate the process description. The model's diagram view is shown in Fig. 2. This system consists of a piping network with 11 pipes and 3 valves, transporting fluid flows from a source on the figure's left side to a sink on the figure's right side. For reasons of clarity, we will limit the processing of this example to basic functions. The full capability of the presented approach in its current state will be shown in section 6 for the example of a district heating network model.

For the first processing step, the `Model` class includes methods to parse the Modelica code of a given file and extract information from its declaration sections as well as from the equation section. These functionalities are of limited scope, however, as they focus only on mapping the model structure into a graph in Python. More complex Modelica features such as extending and redeclaring are not processed by this simple parser. For the component model declarations, the parser extracts data about the component's class, its instance name as well as the coordinates of its graphical representation, which can be read from the corresponding annotation. At the current stage, this step will process only declarations of components that have been selected in advance. This limitation arises from the fact that later in the process, special subclasses are needed to extract relevant information from the simulation results for each type of component.

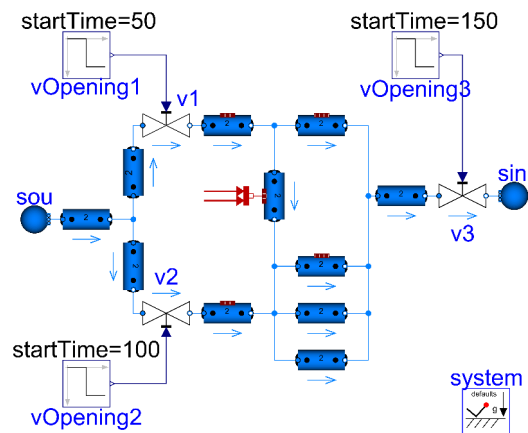


Figure 2. Diagram view of the example model for an incompressible fluid network from `Modelica.Fluid`

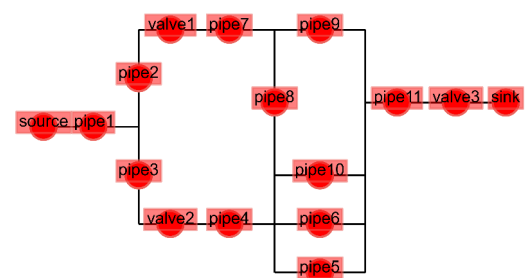


Figure 3. Representation of the example model in a `Model` graph

In the example model, we prepared only for the fluid components to be processed. As a result, sub-models that are not an integral part of the fluid network, like the system model in the lower right corner of Fig. 2 and the control inputs for the valve openings are not taken into account to be part of the `Model` graph. If of special interest, a processing of these sub-models could also be implemented into the presented framework. Yet, for larger fluid networks, this may compromise the clarity of the visualization.

Having set all the graph's nodes according to the relevant model components, the parser returns to the Modelica model file to extract the connection statements. For each connection statement involving those component instances that are represented as a node, an edge is added to the graph accordingly. In order to conserve all relevant graphical information from the Modelica code, the parser also processes the annotations of the connection statements. If a connection in the Modelica code is not drawn directly between two ports as a straight line, the intermediate points given in the annotations are inserted to the graph as separate network nodes. As a result, the graphical representation of the graph will better match the original Modelica model. For the example model shown in Fig. 2, the `Model` graph is displayed in Fig. 3.

Before reading simulation data to the graph, we sug-

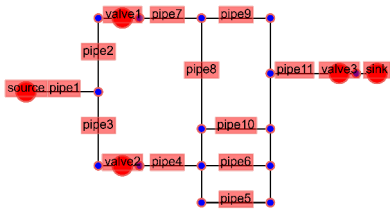


Figure 4. Representation of the example model in a `System` graph. Network nodes are displayed in blue

gest converting the `Model` graph to a `System` graph. This transformation can help to make the visualization more intuitively comprehensible to the user. In this example, we transform the `Model` graph in such a way that the pipes are converted to be edges between network nodes instead of nodes themselves. For this example, we decided to keep the valves as nodes, thus showing both possible pathways of keeping a component type as nodes and converting nodes to edges for the pipes. This could be changed according to the specific application with little effort.

Fig. 4 shows the result of the conversion, with the network nodes marked in a blue color and the pipes being represented by edges. Even though the advantages of this transformation may not be highly significant for this simple example, the use case in section 6 will demonstrate the benefits in the context of a larger pipe network. Furthermore, the distinction between `Model` and `System` graphs allows for more dedicated class definitions with focus on parsing the Modelica file for the `Model` class and focus on visualization for the `System` class.

4 Reading Result Data to Graph

The `System` graph is created as a data structure and template to visualize the dynamic system behavior. In order to read the simulation result data into this structure, the `System` class can access top-level system data directly by making use of result handling methods from the package `ModelicaRes`. For handling result data of the individual components, `System` calls special `Component` classes. A basic `Component` class defines methods for extracting certain data from the result file for a component in a general way. Examples for such methods are `get_mass_flow_rate` or `get_temperature`, which return the time-series of mass flow rate or temperature in the component respectively.

As the identifiers for each component's variables may be different, we extend this general `Component` class for every relevant component type and assign it its own class with specific identifiers and in some cases with special functionalities. In the example of 2, three such classes are needed, i.e. the classes `Boundary`, `Valve`,

and `Pipe`. As extending the `Component` class requires relatively little effort, we prefer this method over the attempt to have only one `Component` class that tries to manage all different component types and their differences.

In terms of processing the data, when adding a node to the `Model` graph, a `Component` object is automatically initialized and attached to the respective node. To this end, `networkX` allows for setting data and objects as attributes to nodes, edges, or the graph itself. In the implemented approach, each object that is attributed to a component's graph representation is also moving from the `Model` into the `System` graph. As a result, all data regarding the component can be accessed by user-friendly methods like the `get_mass_flow_rate` mentioned above for each node and edge. Thus, the object-oriented approach from the Modelica model is followed also in the post-processing by an object-oriented Python implementation.

For components that do not correspond to any Modelica component directly, in some cases the code will assign the object of a neighboring graph element. For example a network node may thus be attributed an instance of the neighboring pipe object, so that it will return the pipe's mass flow rate when queried for such data. In some cases, like for an edge between two network nodes, there may also not be a neighboring object that directly represents a Modelica model component. For this situation, we sometimes prefer not to attach any data to it in order to not give any wrong impression in the visualization. Yet, there is the possibility to interpolate some of this data when the user wants it visualized in a certain way.

When thinking about ways to visualize different kinds of data for various components, some ways of display seem more intuitive than others. As mentioned above, some of the most relevant data to visualize for a thermo-fluid network are mass flow rates, pressures, enthalpies, and temperatures. Often, mass flow rates and temperatures are of special interest. In a non-Modelica context, Köcher (2000) reported a way of visualizing pressures and temperatures in a district heating network at the nodes. Yet, in the `System` class representation shown in Fig. 4, most of the mentioned information concerns the edges rather than the nodes. Thus, the way the edges are drawn in a graph plot are a central part of the visualization. In order to prepare that visualization, we use selected data to calculate edge weights and edge colors to be used in the plotting.

The selection of what values to represent by edge weights and colors is up to the user. The `System` class contains methods for both calculations, that take as arguments the variable that is to be represented, e.g. `temperature` or `mass flow rate`. Based on this selection, the edge weight and color will be calculated and attributed to the corresponding edge. In the case of color representation, a relative value between the mini-

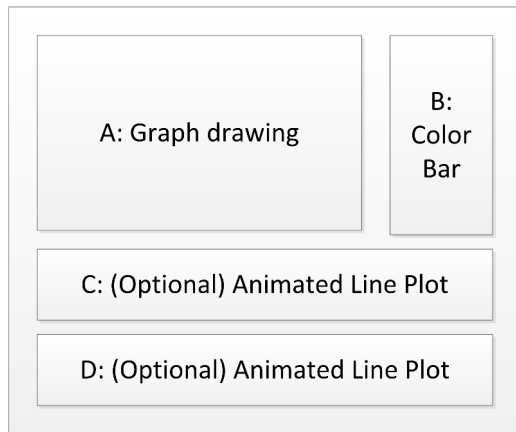


Figure 5. Schematic view of the visualization grid structure

imum and maximum value will be calculated and mapped to a color coding using `matplotlib`'s color-mapping function.

After the graph construction, transformation, and result file handling, the `System` graph will contain all relevant data for visualization. This data structure concept has proven to be user-friendly and efficient, making all data easily accessible and fast to process. The `System` representation as described above presents a compromise between the most intuitive design and strictly following the Modelica model setup. This compromise may be evaluated for each use case and the graph representation adjusted accordingly. This is possible with moderate manual effort, as the object-oriented and graph-based code structure should be reasonably transparent for the user.

5 Visualization of Fluid Flows

In order to keep the visualization output as flexible as possible, we define a framework for sub-plots using `matplotlib`'s grid structure. Fig. 5 illustrates the concept. The only fixed properties are the spaces *A* and *B* that serve as placeholders for the network graph drawing and the corresponding color map. In many cases, one can argue that such a graph drawing has advantages over a multitude of standard 2D line plots. Yet, we do not want to argue that it is inherently always superior to the clarity and simplicity of a line plot. Therefore, any number of line plots can be placed beneath the graph drawing in any number of spaces *C*, *D*, and so on. The `System` class allows the user to name the variables that should be plotted in addition to the graph drawing.

Regarding the graph drawing for space *A* in Fig. 5, the user can select different visualization types. Most times, this will consist of a 2D view recreating the `System` graph as illustrated in Fig. 4, with the edge weights and colors varying according to the preselected variables. For the future, we will also work on 3D plots, where the value of an additional variable can be visualized by

use of a z-axis. This is especially interesting to visualize pressure levels so that mass flows will flow from nodes plotted at greater z-axis levels to those with lesser z-values.

In any case, the Python routine will create a plot following the structure shown in Fig. 5 for every time-step in the simulation result file or for a user-selected period within the simulation time limits. The graph drawing will loop over all nodes and edges, plotting them into space *A* and adjusting their appearance according to data like the node type, edge weight, and edge color stored in the `networkX` graph data structure. For the line plots, the lines will be drawn from the time-step at the beginning of the visualization until the current time-step for this plot. Thus, when the individual static plots are compiled into a video, this will give the impression of a line plot tracking the behavior of the corresponding variable with each time-step.

Returning to the illustrative example introduced with Fig. 2, we can demonstrate the graph drawing part of the visualization output. Unfortunately, as the presented approach directly aims at overcoming the limits of static data plotting, it is hardly possible to show the benefits of an animated visualization in the form of this paper. Therefore, we attempt to mitigate this shortcoming in the paper by using the timeline representation given in Fig. 6. In order to avoid distractions, we limited the display to the plain graph drawing for four steps during the simulation time of 200 s.

There are three changes happening during simulation, namely the closing of valve 1 after $t = 50$ s, the partial closing of valve 2 at $t = 100$ s and the partial closing of valve 3 after $t = 150$ s. The graph drawings show how these changes affect system behavior. After the closing of valve 1, the upper pipe branch is cut off from the flow between the source at left and the sink at right. The partial closing of valves 2 and 3 shows the effect of a reduced mass flow rate in the whole system, depicted by a thinner line thickness for all connections.

In order to better demonstrate the functionalities of the color mapping, we made one slight change to the original model from `Modelica.Fluid`. In the original model, the temperature at the source is kept constant, and the heat source in pipe 8 only has a limited effect on the system as a whole. Therefore, we changed the source temperature to start at 80 °C and decrease linearly until the end of the simulation to 20 °C. This decreasing temperature can be seen in Fig. 6, represented by the changing edge colors. In calculating the edge colors, we used temperature values derived from the average enthalpy between the two fluid ports of a component.

For animating the individual plots in a video, we use the lightweight and freeware software `Images to Video` (Sivic, 2015). This software can be called via a command line interface with all settings saved in an XML file. As these steps can be executed from within the Python environment, the solution requires no effort



Figure 7. Diagram view of the district heating network model

from the user. Also, the user is free to work directly with the individual plots created or use different software to create a video. Still, this part could be improved upon if a Python package for creating the video could be used instead for a more integrated process.

6 Use Case: District Heating Network

In the previous sections, we used a rather academic example to demonstrate the process and functionalities of the presented approach. In this section, we show a use case for which the presented visualization tools were originally developed. We investigate a district heating network that supplies about 120 buildings with heat from one central heating plant. To model this system, we use simplified component models for pipes, the building substations, and the supply. The graphical representation of the system model is shown in Fig. 7. The pipe models calculate a pressure drop depending on the mass flow rate and have a thermal connection to the ground temperature to calculate thermal losses. The building substation models include a control valve, adjusting the mass flow rate according to building heat demand given as a table input. The supply model consists of a simple pump model and an ideal heat source, controlling the network’s supply temperature to a set temperature depending on the outdoor air temperature.

Considering the about 120 buildings, over 200 pipe elements in the supply and return lines, and the loops

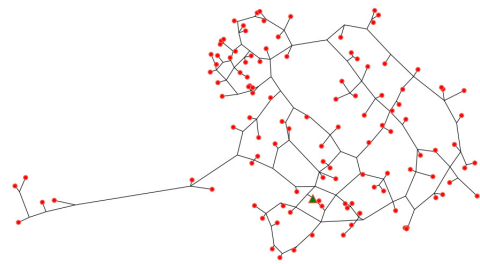


Figure 8. The district heating network’s System graph

in both, the district heating network qualifies as a complex thermo-fluid system. Using only 2D line plots to visualize mass flow rates and temperatures for the entire system can thus be cumbersome. In this context, the presented visualization approach can help to verify and better understand the system behavior of the model.

For the system model, there are two largely identical pipe networks, one for the supply lines from supply plant to the buildings and one for the return lines from buildings to the supply plant. We modeled both these networks, but only used graphical annotations for the Modelica code of the supply lines. Therefore, the return pipes and their connections are not shown in the diagram view of the Modelica model. This leads to a clearer model view, yet makes it even more important to verify the model results in order to ensure that all these connections are correct.

As the Model class processes the Modelica code in terms of declaration statements, connections, and their graphical annotations, the missing graphical annotations lead to the return components neither being represented in the Model nor in the System graph. The resulting System graph for the district heating network is shown in Fig. 8. Nevertheless, the values of the return pipes can be shown in the graph in place of the supply pipes’ values, as the return lines are placed at the same locations as the supply lines. For the data handling of each component, we extend the general Component class and

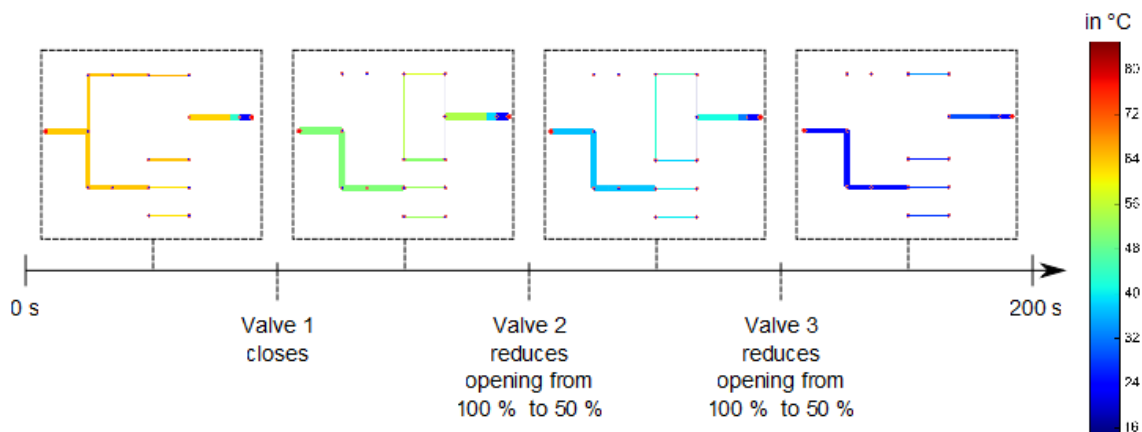


Figure 6. Using graph drawing to visualize system behavior over time

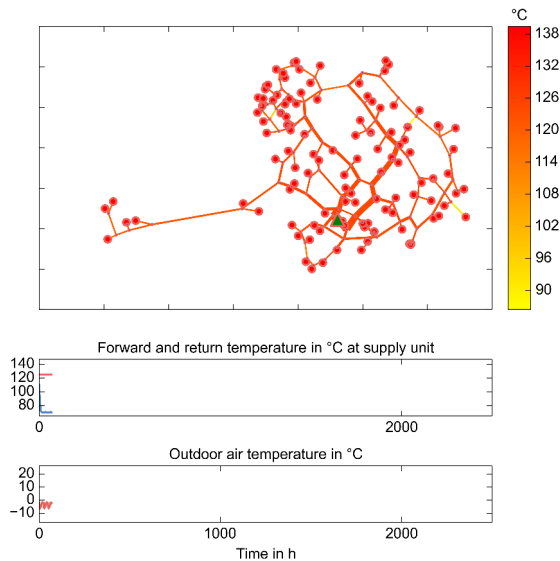


Figure 9. Visualization output for the district heating network at a low-load operation at the beginning of the simulation

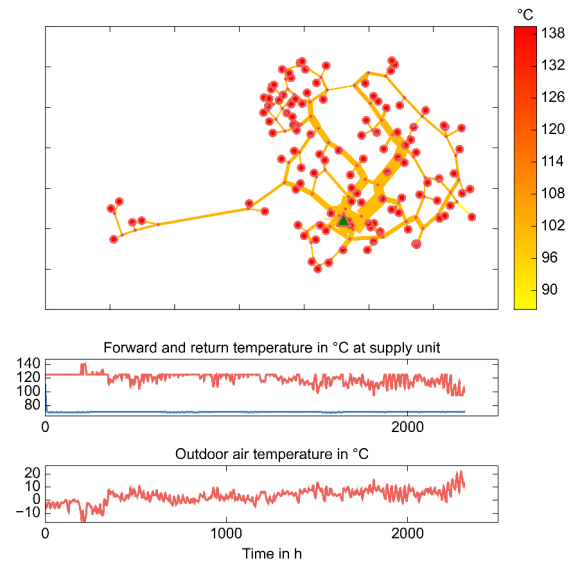


Figure 10. Visualization output for the district heating network at a medium-load operation near the end of the simulation

define the identifiers for different variables in the result file as described in section 3. Thus, it is possible to use a `Pipe` component class that retrieves the supply or the return pipes' data at the user's selection. Similarly, we use a `Supply` and a `Building` class to handle the data of these components.

To demonstrate the visualization approach for this use case, we use a simulation of the district heating network model for a simulation time of 2500 hours with an hourly time-step. Starting at the beginning of the year, this illustrates the first part of the year with significant heat loads. Following the layout of Fig. 5, we use the graph drawing to visualize mass flow rates and the temperatures in the supply lines as well as two line plots. One line plot shows the supply plant's supply and return temperatures and the second line plot shows the ambient outdoor temperature as read from the weather input file.

Again, this paper is limited to show static plots of the visualization for given time-steps. A further application is to compile a video from all the plots to animate the dynamic system behavior. For this demonstration, Fig. 9 shows the state of the system near the beginning of the simulation while Fig. 10 shows the system closer to the end of the simulation. By varying the line width of the pipe connections according to pre-calculated edge weights that depend on the mass flow rate, it is possible to show the mass flow rates for all of the supply lines' over 100 pipe elements in one single plot. Together with the color mapping of water temperatures within the pipe to the color bar given on the upper right, the plots give an impression of the energy flows in the network.

A comparison of Fig. 9 and Fig. 10 illustrates the concept of line plotting in the lower part of the figures. As the line is plotted from the beginning until the current time-step, it gives an impression of monitoring the

selected simulation results when animated into a video. Furthermore, it serves as an indicator of the current time-step even in a static plot and enables the plotting of data that would be difficult to represent by color and line thickness or variables that are not directly part of the thermo-fluid network like the ambient temperature. This enriches the context for the graph drawing visualizing the energy flows in the network.

By visualizing energy flows in the graph drawing, the presented approach can be a useful tool in verifying simulation results. For verification purposes, the main advantage of the graph-drawing based visualization approach over simple line plots is that various system variables are shown together and in context of the system behavior. Furthermore, it would be possible to not only display simulation result variables in such a visualization, but to also display deviations from measurement data, if such data is available.

Once verified, the visualization can also be used as a tool to better understand system behavior and thus assist in planning of the system operation. In real-world thermo-fluid networks, the exact ways the energy flows take is often not known. Especially in district heating networks that include multiple loops and where the pipes are buried in the ground, it can be hard to measure the direction and flow rates of all the pipes. In these cases, as Fig. 10 indicates, the visualization can help to identify main routes of energy flows as well as pipe elements with low flow rates. Yet, to draw conclusions for the operation of the actual system, efforts must be made to verify such observations in the real-world system, as model assumptions and malfunctions in the actual system can lead to deviations between simulation and real-world operation.

7 Conclusions

This paper presents an approach that uses post-processing of Modelica simulation results and graph drawing in order to better visualize the dynamic behavior of complex thermo-fluid networks than standard line plots of individual result variables. Using a graph and attributes for nodes and edges as a data-structure to handle Modelica simulation results has proven a feasible concept, as it can mirror the object-oriented structure of the Modelica model into the post-processing. This allows for a low-maintenance framework that nevertheless offers flexibility for adjustments and options to tailor the visualization output to the specific aims of the visualization and to the requirements of the used models.

Regarding the computational performance, processing the data as well as the `Model` and `System` graphs creates little overhead and takes a few seconds on a standard laptop computer. The time for the plotting will largely depend on the model size, time-step, simulation time, and the required resolution of the output data. Therefore, this part of the process can currently take from a few minutes up to 2 hours for a very high-resolution animation of a large district heating system simulation with small time-steps and a duration of 1 year. Yet, it is likely that the time this part of the process can be efficiently reduced by parallelization of the plotting.

The functionality of the presented approach was demonstrated for a simple example from the Modelica Standard Library as well as for a real-world application of a district heating system model. In this proof of concept, we used line thickness to visualize mass flow rates from one node to another and line colors to indicate temperature levels. Other possible uses include visualizing pipe diameters with line thickness or flow velocities with line colors. Also, we limited our graph drawing to 2-dimensional representations of the system, which resembles the diagram view of the corresponding Modelica models. In this process, parsing the Modelica code for the graphical information in the annotations leads to nodes in the graph with corresponding coordinates. In future work, it will be interesting to visualize certain values in a pseudo-3-dimensional way, where the model representation can stay in the x- and y-axes while simulation result values can be shown on a corresponding z-axis. This is especially promising to visualize pressure levels of supply and return lines for thermo-fluid networks or deviations between simulation results and measurement data.

We argue that the presented approach can be a useful tool in handling the complexity of larger thermo-fluid networks and their dynamic system behavior. On the one hand, the visualization of energy flows and other simulation result data can help modelers to verify their model setups and assumptions. On the other hand, the visualization can be used to inform about relationships and interactions of system components. Yet, drawing con-

clusions from such visualization for the operation and design of actual systems, similar to all aspects of modeling and simulation, requires critical verification of the models used and the results obtained.

Furthermore, the process of visualizing Modelica simulation results introduces methods to parse Modelica code and handle information about model structure and behavior in a Python-based graph structure. For the future, it will be interesting to use these resources for the automated generation and modification of Modelica models. To this end, we are working on a bi-directional work-flow to generate Modelica models for district energy systems from different input data with the `System` graph at the conceptual core. Possible input data includes data from geographic information systems (GIS) or CityGML. In reverse, these models and their results can again be processed by the `System` graph as described in this paper. Thus, the `System` graph can be used as the foundation in an integrated workflow for model generation as well as result analysis and visualization. We think that such an approach has the potential to address handling the complexity of input and output data of large-scale energy system models, which has been identified as one of the key challenges in modeling such systems (Keirstead et al., 2012).

This will hopefully reduce manual effort in modeling complex system like district energy systems and lead to insights from modeling these systems for real-world applications. To this end, we plan to release the developed Python code as an open-source package in the near future. In addition, the Modelica component models for the district heating network modeling will be made available through the open source model libraries `AixLib`¹ and its contributions to the Annex 60 library², which is a joint effort within the International Energy Agency's Annex 60 programme.

Acknowledgment

We gratefully acknowledge the financial support by BMWi (German Federal Ministry of Economic Affairs and Energy), promotional reference 03ET1260A.

References

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 Modelica Association, editor, *Proceedings of the 5th International Modelica Conference*, pages 631–640, 2006.

Kevin Davies. ModelicaRes python package, 2015. URL <http://kdavies4.github.io/ModelicaRes/>.

¹<http://github.com/RWTH-EBC/AixLib>

²<https://github.com/iea-annex60/modelica-annex60>

- Roel De Conick. awesim python package, 2015. URL <https://github.com/saroele/awesim>.
- Arash M. Dizqah, Alireza Maheri, Krishna Busawon, and Peter Fritzson. Standalone DC microgrids as complementarity dynamical systems: Modeling and applications. *Control Engineering Practice*, 35:102–112, 2015. ISSN 09670661. doi:10.1016/j.conengprac.2014.10.006.
- Tingting Fang and Risto Lahdelma. State estimation of district heating network based on customer measurements. *Applied Thermal Engineering*, 73(1):1211–1221, 2014. ISSN 13594311. doi:10.1016/j.applthermaleng.2014.09.003.
- Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.
- Matthias Hellerer, Tobias Bellmann, and Florian Schlegel. The DLR Visualization Library - recent development and applications. In *the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, Linköping Electronic Conference Proceedings, pages 899–911. Linköping University Electronic Press, 2014. doi:10.3384/ECP14096899.
- Christoph Höger, Alexandra Mehlhase, Christoph Nytsch-Geussen, Karsten Isakovic, and Rick Kubiak. Modelica3D - platform independent simulation visualization. In Modelica Association, editor, *Proceedings of the 9th International Modelica Conference*, pages 485–494, 2012. doi:10.3384/ecp12076485.
- John D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, May-Jun 2007.
- James Keirstead, Mark Jennings, and Aruna Sivakumar. A review of urban energy system models: Approaches, challenges and opportunities. *Renewable and Sustainable Energy Reviews*, 16(6):3847–3866, 2012. doi:10.1016/j.rser.2012.02.047.
- Ralf Köcher. *Beitrag zur Berechnung und Auslegung von Fernwärmenetzen*. PhD thesis, Technische Universität Berlin, Berlin, 2000. URL <http://d-nb.info/960177469/34>.
- LBL-SRG. BuildingsPy python package, 2015. URL <https://github.com/lbl-srg/BuildingsPy>.
- Jaromir Sivic. Images to video v4.0, 2015. URL <http://en.cze.cz/Images-to-video>.

Reuse of Physical System Models by means of Semantic Knowledge Representation: A Case Study applied to Modelica

Elena Gallego¹, Jose María Álvarez-Rodríguez¹ and Juan Llorens¹

¹ Knowledge Reuse Group,
Department of Computer Science and Engineering,
University Carlos III of Madrid, Spain,
{elena.gallego, jmalvarez, llorens}@kr.inf.uc3m.es

Abstract

This paper presents the design and development of a solution to store and reuse physical system models by indexing and retrieving their content and metadata. To do so, a mapping between the representation modelling language and a semantic-based representation model (Relationship-RSHP) is defined. More specifically, electrical circuits designed in Modelica have been mapped to RSHP. A two-step process has been designed and implemented to parse Modelica artifacts and index the contents into a system knowledge repository. Afterwards, a case study has also been conducted to compare text vs. concept based information retrieval processes. A dataset of 25 electrical circuits and a set of 30 queries have been designed to extract precision and recall metrics assessing that the presented approach improves the retrieval of Modelica artifacts. As main conclusion, it is possible to state that a domain specific technology such as RSHP for knowledge representation can help the management of Modelica artifacts as knowledge assets.

Keywords: Information Representation, Physical System Models, Modelica Language, Model Reuse, Knowledge Reuse.

1 Introduction

Cyber-physical systems (CPS), a set of collaborative computational resources controlling physical entities, are considered “*the next computing revolution*” (Rajkumar et al. 2010) (K.-D. Kim and Kumar 2012). The design and deploy of these systems is currently based on the 5C architecture (connection, conversion, cyber, cognition, and configuration). Physical system models are designed at different levels of abstraction to analyze and study the mathematical equations that govern the CPS under different excitation configurations.

To do so, software tools (Fritzson 2015) supporting physical modelling languages are used to design and run the simulations that represent the physical system model behavior. During this stage of design and development a good number of logical artifacts are generated. In this context and with the aim of easing the development of the 5C architecture, software developing environments

usually provide libraries of reusable components (M. Kim et al. 2010) through application patterns (Choi et al. 2013) and other techniques. These components are commonly represented in a particular modelling language and tagged with a predefined set of metadata properties that can only be accessed from the same development environment that produced them.

In order to reuse a component, the first step lies on the capability to search for them through a traditional interface, filtering the potential results depending on keywords or fixed values in the metadata fields.

Assuming that a physical system model in some modelling languages, such as Modelica, is a software artifact, it is possible then to apply the well-known techniques for information and software reuse (Jacobson, Griss, and Jonsson 1997) (Karlsson 1995). Reuse of information and software may have the potential of increasing productivity of engineers, improve quality and create a cost efficient development environment for cyber-physical systems.

However, the systematic support of reuse is affected by technical and non-technical issues (Smolárová and Návrat 1997):

1. Economical, organizational, educational or psychological issues and
2. Lack of standards to represent all software artifacts, lack of reusable component libraries or appropriate tools for boosting reuse among tools.

In the context of technical issues, those considered in this paper, the classical principles of (software) reuse: abstraction, selection, specialization and integration, can be found in a very good number of works (Jacobson, Griss, and Jonsson 1997) (Karlsson 1995) (McIlroy 1969). In particular, *abstraction* (management of the intellectual complexity of an artifact) can be considered the essential feature for any reuse technique in order to specify when an artifact could be reused and how to reuse it. *Selection* refers to the discovery of artifacts covering from the representation and storage to the classification, location and comparison. *Specialization* consists on the set of parameters and transformations required to reuse an artifact, while *integration* refers to the capability of systems to communicate, collaborate and exchange data.

Thus, the reusability factor of artifacts will directly depend on how they are abstractly described, how they can be selected and specialized for reuse, and how they will integrate in the new complete system.

Currently, knowledge management has gained momentum in the software domain as a means to elevate the meaning of the implicit knowledge represented into software pieces. Software is becoming a commodity that is embedded in any work product or business process, being a new kind of intellectual asset that can be used to reduce costs and time to market by generating competitive advantage.

In this light, knowledge management techniques (Nonaka and Takeuchi 1995) can be applied to capture, structure, store and disseminate software-based artifacts to directly support the aforementioned software reuse principles of selection and integration. However, the selection of a proper knowledge management mechanism is still an open issue (Hull and King 1987) due to the fact that a suitable representation model can be reached in several ways.

In the context of cyber-physical systems development, physical system models seem to be a good candidate to take advantage of knowledge management and reuse techniques. Based on this concept, the Modelica modelling language (Fritzson and Engelson 1998) (Fritzson 2015) provides a comprehensible model data structure (Schamai, Fritzson, and Paredis 2013) in which it is possible to develop, design and run simulations.

However, there is much more at stake than the simple representation in a modelling language. Physical systems are represented by equation systems or by graphical models that represent their behavior. This valuable information must be organized and stored to be able to provide high-accurate information retrieval processes. One of the main challenges emerges from the complexity to transform physical systems into a logical structure that can be modeled and understood by knowledge management tools.

Semantic knowledge representation models appeared around year 2000 to cope with complex information representation problems. The most representative example of them can be Resource Description Framework (RDF) (Hayes 2004) and RSHP (pronounced “arship”) (Llorens, Morato, and Genova 2004). RDF was created from the beginning to cope with web information management while RSHP’s main goal was to represent information from all industrial work-products.

In order to overcome the existing limitations on reusing physical system models knowledge, a mapping between the Modelica modelling language and the RSHP information representation model is defined and implemented (Modelica2RSHP). Due to the intrinsic RSHP capabilities, it is possible to represent any kind of information such as textual descriptions, design models,

code or even any piece of relation data under the same schema. A tool implementation for managing industrial work products has been developed by The Reuse Company (The Reuse Company Inc. 2014), named knowledgeMANAGER, enabling the possibility of applying knowledge management techniques to engineering domain.

As motivating example, Figure 1 shows a simple amplifier circuit comprising different electrical elements. This block could certainly be reused in different cyber-physical systems. However, in order to allow reuse the proper mechanisms must be provided to represent the elements and relationships within the circuit (metadata and contents), to store such elements in a repository, to define a retrieval algorithm that would allow the identification of physical models by content and to retrieve the block according to different queries. For instance, an engineer should be able to look up this circuit, see Figure 1, by expressing the next query: “Give me all electrical circuits that contain a sine voltage source directly connected to an operational amplifier by a 20kΩ resistor”. In current Modelica environments, these tasks are hard to accomplish since they were not designed for these purposes. Advanced regular expressions could be a solution but an approach taking advantage of describing elements and relationships can really improve the retrieval of Modelica artifacts boosting the reusability factor of existing physical system models.

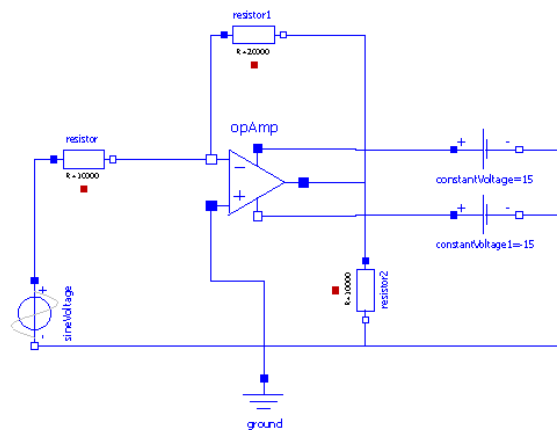


Figure 1. Simple Amplifier circuit which uses an operational amplifier (see example in Electrical-Analog circuits in OpenModelica).

2 Physical system models as software artifacts

Software reuse (Smolárová and Návrát 1997) as a discipline has been widely studied and surveyed from different perspectives. Reuse depending on software metrics and models (Frakes and Terry 1996), reuse of software libraries (A. Mili, Mili, and Mittermeir 1998), software repositories (Guo and others 2000), components in the industry (Land et al. 2009), success

factors (Basili and Rombach 1991) and reuse in software product lines (Thüm et al. 2014). In all of them, the different authors have explored and classified the mechanisms to store and retrieve software assets. One of the main conclusions in these studies is that successful reuse will come with sophisticated software components storage, representation and retrieval techniques. In this light, the authors in (Guo and others 2000) define a set of orthogonal attributes and six broad classes of methods for software reuse. They also establish criteria (technical, managerial and human factors) to assess and compare classes of methods for software reuse.

Other very relevant works have been focused on applying control engineering techniques (H. Mili 2002) for software reuse. Although some of good experiences have been reported (Tracz 1995), success and failure facts outlined in (Morisio, Ezran, and Tully 2002) and (Desouza, Awazu, and Tiwana 2006) are still open. This situation of software reuse is becoming critical in cyber-physical systems where the time to design, develop and deploy a system is more complicated due to the collaboration with other software and hardware components.

2.1 Physical system models sharing and reuse

When thinking about models reuse, engineers have to deal with the underlying information of a shared model and its relation with the design. Human experience is important to correctly understand, share or reuse models efficiently, while machines usually fail because of the tacit knowledge involved.

In (Winsberg 2001) the authors present a semantic driven design reuse for a 3D scene designed by computing the properties while modelling and enabling the system to recognize similar types by a vertex statics based algorithm.

As (Groza et al. 2009) outlines, over 20 billion CAD models exist with similar geometric aspects. Currently, indexers use alphanumeric numbers with different formats for each group. The developer could be able to design new models based on existing ones and reuse their similar components. More than 75% of new models design could be reused from previous models ensuring that the model fulfills the functionality for which it has been designed.

After this brief analysis, there are many technical problems (including data protection or copyrights) to create agreed knowledge-based representations such as ontologies that can ease the sharing and reuse of physical system models produced by different tools.

One of the most necessary elements, once a common knowledge representation is defined, is to have a good search engine supported by domain knowledge. This is the main goal for future works, to be able not only to store physical system models, but also to look for similar

models and retrieve their information using concepts and relationships.

Functional Mock-up Interface (FMI) is described in (Otter, Blochwitz, and Arnold 2013) as a solution to model sharing and reuse. FMI allows to work with different simulation environments, as Modelica, Simulink and SIMPACK just in one interface to enhance model sharing avoiding incompatibilities.

Using current design tools it is possible to get both analytical and visual representation for every developed physical system model.

The analytical information describes the physical laws that model the system while the visual representation usually shows them graphically. Visual information represents a simplified view of the world that the system is modelling. When thinking about reuse of physical models, the approach should be to work with the analytical information, because of the knowledge contained. The analytic part of a model represents the different behaviors that could be in the real world for many configurations.

That is why; the choice made in this work is to index the analytical information of any physical system model, which can be complemented by graphical information when retrieving it easing the understanding of the underlying knowledge.

3 Physical System Models

The complex world where we live has the inherited characteristic to be governed by physic laws, which humans continuously try to control. Every physical system that engineers want to better understand has elements that behave according to a set of physical laws (Winsberg 2001).

Physical systems models represent the reality by means of relationships between physical and mathematical theories and their effect in the reality. There exist many ways to design physical system models but, almost all of them, are constructed under the same theories.

Therefore, if we are aware of the elements that define the system and the physical laws that govern it, we have the required information of the physical system model, in order to get the knowledge, with different abstraction levels, which can be used in other processes or projects.

Physical system models can be as complex as the reality they represent, thus, it is needed to clearly define the purpose of the model in order to get a reasonable result.

The goal, when modelling physical systems, is to get a mathematical representation of the system's behavior in terms of its variables. Depending on the nature of the system, electrical, mechanical or thermal, the system variables change. Despite of the differences, a common concept between the disciplines is energy, so it is possible to design the physical components of the system as energy manipulators (Wellstead)

Physical system models are built to represent the real world where the model is going to be used and its response to particular stimuli. The needs to create physical system models are described in (Valášek et al. 2003) as the real world system, the question to be answered by the simulation of the model, and the interpretation of the output is the solution.

3.1 Physical systems modelling environments

There are many models design environments that offer different capabilities depending on the domain.

Modelica-based modelling and simulation environments such as Dymola (Dempsey 2006), OpenModelica (Asgha and Tariq 2010) or JModelica (Åkesson et al. 2010), are examples of integrated development environments that make easier the visual development of models in domains such as: electric, mechanic or thermodynamic.

More specifically, the Modelica language is an object-oriented programming language that allows physical systems modelling. Models can be expressed by differential, algebraic and discrete equations. Modelica allows reuse and share models by reducing the modelling effort (Martin-Villalba, Urquia, and Dormido 2008). Nevertheless, the knowledge management capabilities of these environments are restricted as it has been outlined in the introduction.

4 Knowledge representation of Physical System Models

In order to provide the proper knowledge management services for cyber-physical systems, it is necessary to select an adequate knowledge representation paradigm. Obviously, different types of knowledge require different types of representation (Davis, Shrobe, and Szolovits 1993) (Groza et al. 2009). In this light,

expressions, rule-based systems, regular grammars, semantic networks, object-oriented representations, frames, intelligent agents or case-based models, to name just a few, are some of the main approaches to information and knowledge modelling.

More specifically, knowledge management also implies the standardization of data and information, that is, any block of information must be structured and stored for supporting other application services.

In this context, two main approaches can be highlighted: 1) the ISO 10303-STEP (“Standard for the Exchange of Product model data”), is a standard for the computer-interpretable representation and exchange of product manufacturing information and 2) the Open Services for Lifecycle Collaboration (Ryman, Hors, and Speicher 2013) (OSLC), an OASIS standard, that is seeking new methods to easily integrate System Engineering tools and build an ideal development and operations environment with special focus on interoperability.

Although both approaches represent very relevant actions to standardize and provide interoperable environments for developing complex systems, they do not directly define a knowledge model (Alvarez-Rodríguez et al. 2015) for representing metadata and contents of work products and artifacts. Besides, it has been demonstrated that the retrieval of information resources does not imply the need of any underlying logic formalism but a powerful framework for expressing concepts and relationships. Due to this fact and previous experiences (Alvarez-Rodríguez et al. 2015), the RSHP universal knowledge representation model has been selected as meta model to semantically describe the elements and relationships that can be found in a physical system model.

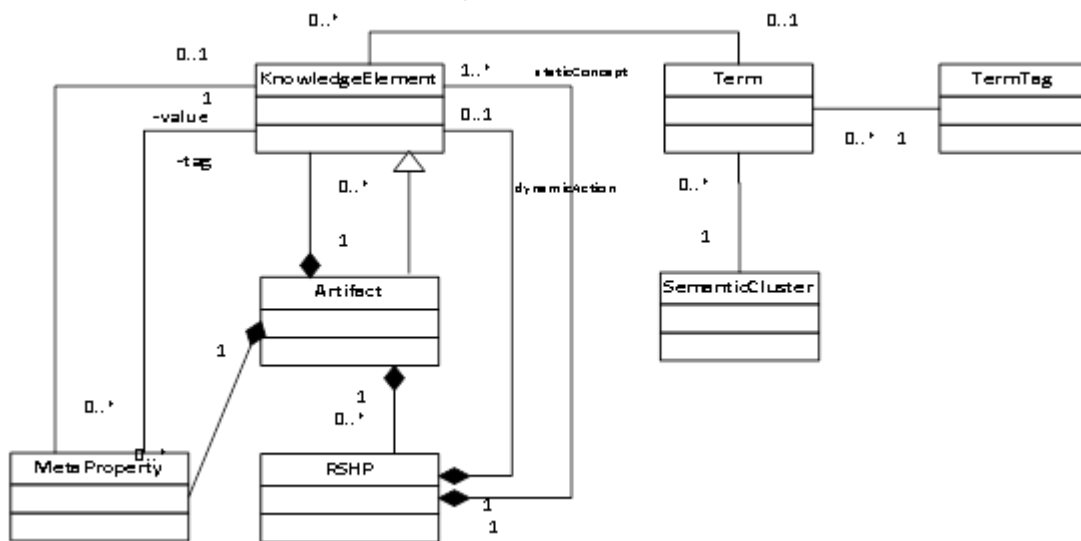


Figure 2. The RSHP representation model in UML.

4.1 RSHP in a nutshell

The RSHP universal knowledge representation model (Llorens, Morato, and Genova 2004), see Figure 2, is based on the ground idea that whatever information can be described as a group of relationships between concepts forming a conceptual graph. For example, Entity/Relationship data models (Chen 1976) are certainly represented as relationships between entities, processes can be represented as causal/sequential relationships between sub-processes, UML (Unified Modelling Language) or SysML meta models can also be modeled as a set of relationships between meta model elements, etc. Furthermore, free text information can certainly be represented as relationships between terms by means of the same structure. To represent human language text, a set of well-constructed sentences, including the *subject + verb + predicate* (SVP) should be used. The SVP structure can be then considered as a relationship typed V between the S and the predicated P. RSHP includes a repository model to store information and relationships with the aim of reusing all kind of knowledge chunks. The RSHP formal representation model, see Figure 2, is based on the following principles:

- The main description element is the relationship since it is the element in charge of linking knowledge elements.
- A Knowledge Element (KE) is an atomic knowledge brick that appears into an artifact and that is linked by one or more relationships with other KEs, to build information. It is defined by a concept, and it can also be an artifact (an information container found inside a wider artifact). A concept is represented by a normalized term (a keyword coming from a controlled vocabulary, or domain). Artifacts are knowledge containers of KEs and their relationships.

In RSHP, the simple representation model for describing the content of whatever artifact type (requirements, risks, models, tests, maps, text docs or source code) should be:

RSHP representation for artifact

$$\alpha = i_{\alpha} = \{(RSHP_1), (RSHP_2), \dots, (RSHP_n)\}$$

where every single RSHP is called RSHP-description and must be described using KE.

One important consequence of this representation model is that there is no restriction to represent a particular type of knowledge. Furthermore, RHSP has been used as underlying information model to build general-purpose indexing and retrieval systems, domain representation models (Díaz et al. 2005), quality assessment of requirements and knowledge management tools such as knowledgeMANAGER (The Reuse Company Inc. 2014).

4.2 Mapping the Modelica language to RSHP

The use of Modelica as language for modelling complex physical systems is gaining momentum in the industry domain (Samlaus and Fritzson 2015). On the other hand, RSHP has been used for a long time in the Systems Engineering discipline for knowledge management. Given this situation, a strategy to map Modelica physical system models to RSHP must be defined. To do so a direct mapping is defined to perform simple transformations and to provide a basis for defining and comparing more complex transformations.

In order to design this direct mapping, both models are represented using the commonly defined abstract data types set and list. The definitions follow a type-as-specification approach (Schamai, Fritzson, and Paredis 2013); thus models are based on dependent types that can also include cardinality. More specifically, Table 1 and Table 2 show both specifications as a kind of regular tree grammars that can be used to specify a rule-based transformation between two grammars (denotational semantics). Thus, a transformation between a partial set of production rules of the Modelica language and RHSP can be defined as a function, $Modelica2RSHP$, that takes the Modelica grammar (v3.2), $G_{Modelica}$, a valid Modelica model, $Modelica_k$, the RSHP grammar G_{RSHP} and a set of direct mapping rules, $M_{Modelica2rshp}$ (see Table 3 where sub-indexes refer to attributes and relationships of the elements), to generate a valid $RSHP_{graph}$.

$$Modelica2RSHP: G_{Modelica} \times Modelica_k \times G_{RSHP} \times M_{Modelica2rshp} \rightarrow RSHP_{graph}$$

Table 1. Selected Production rules of the Regular Tree Grammar of Modelica: $G_{Modelica}$

(1)	<code>class_definition ::= class_prefixes class_specifier</code>
(2)	<code>class_prefixes ::= (model)</code>
(3)	<code>class_specifier ::= long_class_specifier </code>
(4)	<code>short_class_specifier</code>
(5)	<code>long_class_specifier ::= IDENT string_comment composition end IDENT extends IDENT [class_modification]</code>
(6)	<code>string_comment composition end IDENT</code>
(7)	<code>short_class_specifier ::= IDENT "=" base_prefix name [array_subscripts] [class_modification] comment IDENT "=" enumeration "(" ([enum_list] ":") ")" comment</code>
(8)	<code>component_clause ::= type_prefix type_specifier [array_subscripts] component_list</code>
(9)	<code>type_specifier ::= name</code>
(10)	<code>name ::= ["."] IDENT {"." IDENT }</code>
(11)	<code>component_list ::= component_declaration { "," component_declaration }</code>
(12)	<code>component_declaration ::= declaration [condition_attribute] comment</code>
(13)	<code>declaration ::= IDENT [array_subscripts] [modification] ->KE Term</code>

```

(14) connect_clause ::= connect "("
    component_reference "," component_reference
    ")"
(15) component_reference ::= ["."] IDENT
    [array_subscripts] {"." IDENT
    [array_subscripts] }
    
```

Table 2. Regular Tree Grammar of RSHP: G_{RSHP}

```

(1) Artifact ::= (Set(RHSP), MetaProperty{0,*})
(2) RSHP ::= (Subject, Verb, Object, Semantics)
(3) Subject ::= KE {0,1}
(4) Verb ::= KE {0,1}
(5) Object ::= KE {0,1}
(6) KE ::= (Term {0,1}) | Artifact
(7) Term ::= (lexicalForm, languageTag, TermTag)
(8) TermTag ::= lexicalForm
(9) MetaProperty ::= (Tag, Value)
(10) Tag ::= {KE, lexicalForm}
(11) Value ::= {KE {0,1}, lexicalForm {0,1}}
(12) SemanticCluster ::= (Term)
    
```

Table 3. Set of mapping rules $M_{Modelica2rshp}$ to transform Modelica physical system models into RSHP

```

(1) class_definition ::= Artifact
(2) class_prefixes ::= MetaProperty
    (Tag="type", Value="model")
(3) class_specifier ::= long_class_specifier |
    short_class_specifier
(4) long_class_specifier ::=
    Artifact(physical_name=IDENT)
(5) short_class_specifier ::=
    Artifact(physical_name=IDENT)
(6) component_clause ::= type_prefix
    type_specifier [ array_subscripts ]
    component_list
(7) type_specifier ::= name
(8) name ::= SemanticCluster (Term=IDENT)
(9) component_list ::= component_declaration {
    "," component_declaration }
(10) component_declaration ::= declaration [
    condition_attribute ] comment
(11) declaration ::= KE (Term = IDENT)
(12) connect_clause ::= RSHP(KE, KE, KE, KE)
(13) component_reference ::= KE (Term = IDENT)
    
```

Although, the presented mapping does not cover all production rules in $G_{Modelica}$, it is correct since only valid Modelica and RSHP models will be accepted and generated.

4.3 Implementation details

In order to implement the mapping rules presented in Table 3, a stepwise process has been carried out. Taking into account that RSHP and its underlying technology (the CAKE API¹) are implemented in the .NET platform and considering the diversity of Modelica parsers, we selected the option of building the JModelica sources (Java) for Modelica version 3.2.

More specifically, the last JModelica sources² were checked out (January 2015) and built using Apache Ant for Java. Afterwards, a JAR (Java Archive) analyzer tool³ was used to extract the dependencies between the different Java libraries and to generate a script that transformed the required Java libraries to .NET DLLs (Dynamic-link library).

This approach was enough to demonstrate the possibility of integrating a Modelica parser in the .NET platform. Thus, it is possible now to offer a universal information representation model to index and retrieve physical system models metadata and contents.

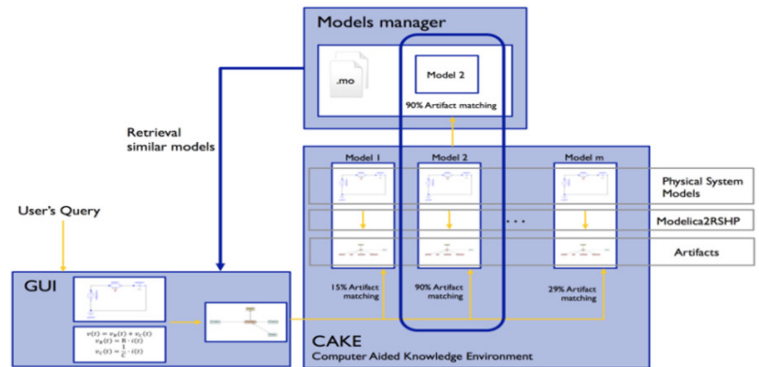


Figure 3. Process to index, search and retrieve a physical system model.

These DLLs are then interpreted in .NET through the IKVM⁴ (a Java interpreter for this platform) providing a port and implementation of the Modelica parser. Finally, this set of .NET libraries are used to implement the set of mapping rules in Table 3 and to connect to the CAKE API as Figure 3 shows. Moreover, the knowledgeMANAGER tool can be used to manage all the generated artifacts, see Figure 4.

¹ The CAKE (Computer Aided Knowledge Environment) API (Application Programming Interface).

² <http://trac.jmodelica.org/browser/trunk/Compiler/ModelicaCompiler>

³ <https://code.google.com/p/jar2ikvmc/>

⁴ <http://www.ikvm.net/>

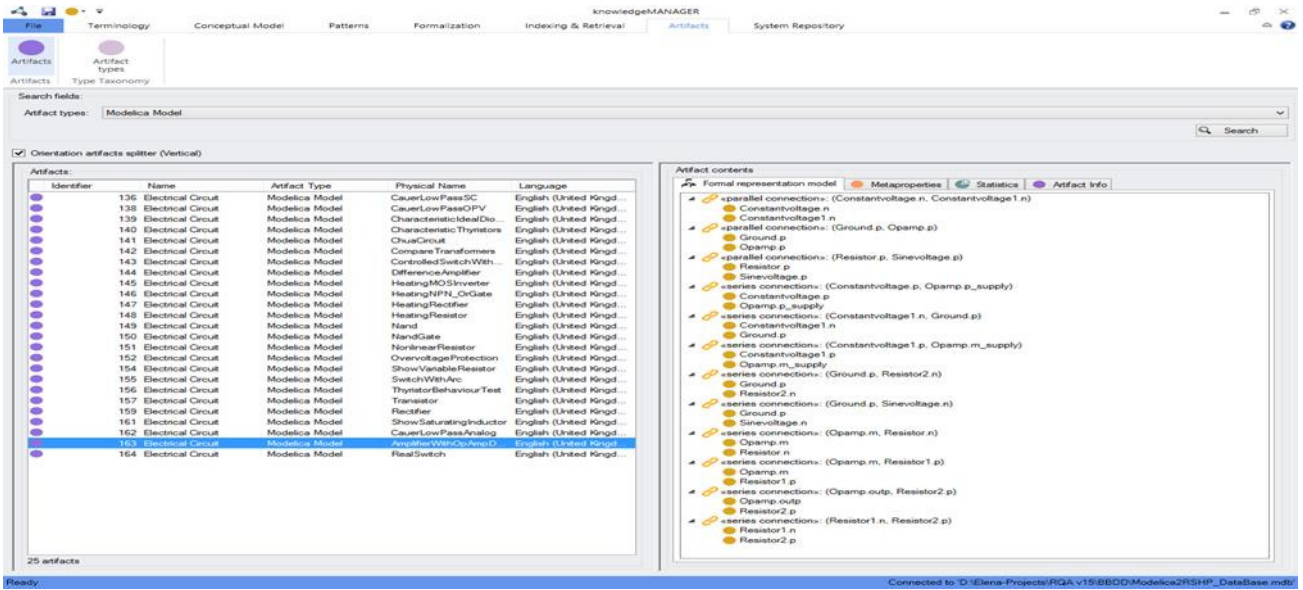


Figure 4. Representation of the physical system models in knowledgeMANAGER

5 Case Study: Indexing and retrieval of Modelica physical system models

To illustrate the approach for reusing physical models, a case study based on the comparison of precision and recall measures of the two approaches to retrieve physical system models (OpenModelica vs knowledgeMANAGER) is presented below.

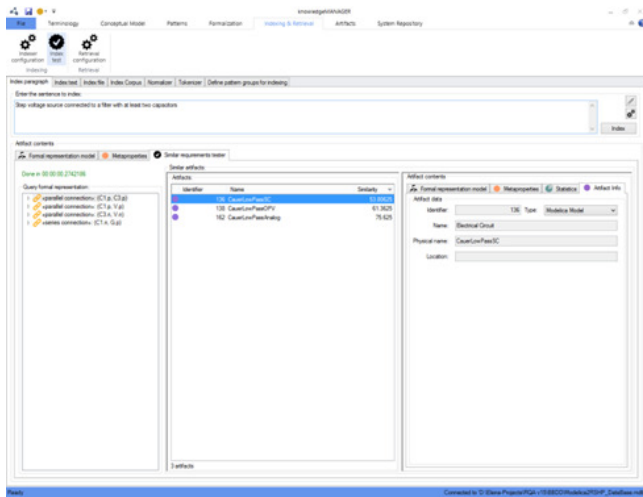


Figure 5. Example of physical system model retrieval in knowledgeMANAGER.

5.1 Research design

One of the main stages in a reuse process consists on looking up the proper artifacts according to a set of preferences or query. This can be interpreted as a search system in which given a query (text-based or even a target model) and a set of resources (a set of physical models), it is necessary to establish which are the best

models that match the input query. To do so, the following steps will be carried out:

3. Design a domain-based vocabulary, O , to represent the concepts and relationships that will be used to represent physical models. In this case, the built in domain ontology in the knowledgeMANAGER has been used. It is actually a taxonomy comprising three main entities: System, Subsystem and Component and hierarchy relationships (*part-of*, *is-a*, *broader/narrower*).
4. Define a test dataset of physical models specifications $D = \{d_1, d_2, \dots, d_k, \dots, d_n\}$. To do so, the public dataset of electrical circuits available in OpenModelica has been selected. This dataset comprises 25 physical system models for electrical circuits that have been also indexed in knowledgeMANAGER, see Figure 4.
5. Define a set of queries and expected results, Q where each query q_k will return a set of physical models D_k . To do so, a random walk process on top of the dataset D has been implemented to automatically generate search queries based on the combination of the different elements that can be found in a circuit (between 1-5). Afterwards, a panel of three experts has validated the expected circuits for every query, see Table 4.

$$Q = \{(q_1, D_1), (q_2, D_2), \dots, (q_k, D_k) \dots, (q_n, D_n)\}.$$

6. Run the indexing and retrieval processes implemented on top of the knowledgeMANAGER APIs and the OpenModelica editor. See an example in Figure 5.

7. Extract measures of precision (P), recall (R) and the $F1$ score (the harmonic mean of precision and recall) making a comparison of the expected and generated results. Being $P = \frac{tp}{tp+fp}$, $R = \frac{tp}{tp+fn}$ and , $F1 = \frac{2 P * R}{P + R}$ where given a target dataset of physical

models, D , and a query q_k which expected results is the set D_k :

- tp (true positive) is the number of physical models in D_k that have been retrieved and are in D ,
- fp (false positive) is the number of physical models in D_k that have been retrieved and are not in D ,
- tn (true negative) is the number of physical models in D_k that have not been retrieved and are not in D and
- fn (false negative) is the number of physical models in D_k that have not been retrieved and are in D .

Table 4. Set of queries to search for physical system models.

Q	Human-based query
q_1	Step voltage source with an RLC filter
q_2	LC filter with any kind of voltage source
q_3	Step voltage source connected to a filter with at least two capacitors
q_4	Step voltage source and operational amplifier
q_5	Comparator operational amplifier
q_6	Diode connected to a sine voltage source
q_7	Ideal Operational amplifier integrator
q_8	Rectifiers with ideal diodes
q_9	Sine voltage source connected to a load by a diode
q_{10}	Sine voltage source connected to a load by two ideal thyristors
q_{11}	Sine voltage source connected to a load by one ideal thyristor
q_{12}	Circuits with thermal resistor and LC filter
q_{13}	Sine voltage source connected to a potentiometer (variable resistor) before a RC filter
q_{14}	Sine voltage source connected to a potentiometer (variable resistor)
q_{15}	Rectifiers with inductances to any load
q_{16}	Inductance filter to a sine voltage source
q_{17}	Sine voltage source connected to a potentiometer to supply a resistive load
q_{18}	Circuits with sine voltage source and a variable resistor
q_{19}	Sine voltage source connected to a resistor
q_{20}	Constant voltage source connected to a LR filter by a switch
q_{21}	Constant voltage source connected to a load by a switch
q_{22}	Sine voltage source and operational amplifier
q_{23}	Simplified transformer connected to a resistive load by resistors
q_{24}	Simplified transformer connected to a resistive load by inductors
q_{25}	Ideal transformer connected to a sine voltage source
q_{26}	switch controlled by a sine voltage source
q_{27}	Sine voltage source with an RLC filter
q_{28}	Sine voltage source connected to a transistor
q_{29}	Circuit whit thermal conductor and heat capacitor
q_{30}	Sine voltage source connected to a capacitive load

8. Check the robustness of the comparison by performing statistical hypothesis testing.

5.2 Results and Discussion

Table 5 shows the metrics of precision, recall and the F1 measure of the different executions. The first column corresponds to the query identifier; the next three columns contain the metric values when the OpenModelica search capabilities are used to look up circuits. After that, the second experiment shows the metric values when the presented approach implemented on top of knowledgeMANAGER is executed. According to the results, it seems clear that the presented approach is better than the results provided by OpenModelica, as Figure 6 depicts. The main reason of this behavior is due to the fact that the presented approach can take advantage of exploiting semantic relationships (knowledgeMANAGER) while the text-based approach (OpenModelica) can only perform string comparisons.

Nevertheless, the precision values can be improved and higher-values would be expected in both approaches. In the case of knowledgeMANAGER, this is because of the detail of the query, when it has more components to compare, the precision is higher. The tool prefers not to return false positives keeping precision higher.

On the other hand, a statistical hypothesis testing has been carried out to demonstrate if results will vary depending on the type of method or tool used to search physical models. To do so, a comparison of the precision values of both tools and approaches has been formulated through the next hypotheses:

H_0 : There is no change in the calculation of precision when searching using OpenModelica or knowledgeMANAGER.

H_1 : There is change in the calculation of precision when searching using OpenModelica or knowledgeMANAGER.

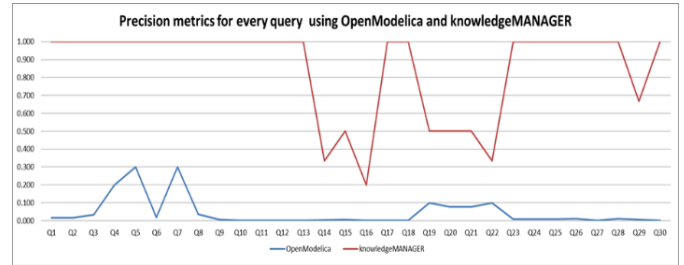
In order to run the statistical hypothesis testing, the F-Test with alpha 0.05 has been carried out to ensure that variances are unequal (there is statistical significance). After that, the t-Test of two-sample assuming unequal variances has been performed with alpha 0.05 to assert whether H_0 is rejected or not. According to Table 6, H_0 can be rejected, since the t Stat is less than “-t Critical (two tail)”. In conclusion, the knowledgeMANAGER tool method exploiting semantic relationships can improve in terms of precision the problem of retrieving the proper physical system models.

Table 5. Precision and recall metrics for a retrieval process in OpenModelica and knowledgeMANAGER.

Q	OpenModelica		knowledgeMANAGER			
	P	R	F1	P	R	F1
q ₁	0.017	0.500	0.032	1.000	0.080	0.148
q ₂	0.017	1.000	0.033	1.000	0.040	0.077
q ₃	0.034	1.000	0.066	1.000	0.120	0.214
q ₄	0.200	1.000	0.333	1.000	0.080	0.148
q ₅	0.300	1.000	0.462	1.000	0.083	0.154
q ₆	0.018	1.000	0.035	1.000	0.000	0.000
q ₇	0.300	1.000	0.462	1.000	0.083	0.154
q ₈	0.036	1.000	0.069	1.000	0.042	0.080
q ₉	0.006	0.500	0.012	1.000	0.042	0.080
q ₁₀	0.000	0.000	N/A	1.000	0.000	0.000
q ₁₁	0.000	0.000	N/A	1.000	0.000	0.000
q ₁₂	0.000	1.000	0.000	1.000	0.000	0.000
q ₁₃	0.002	1.000	0.004	1.000	0.040	0.077
q ₁₄	0.004	1.000	0.008	0.333	0.045	0.080
q ₁₅	0.006	0.500	0.012	0.500	0.043	0.080
q ₁₆	0.000	0.000	N/A	0.200	0.056	0.087
q ₁₇	0.002	1.000	0.004	1.000	0.040	0.077
q ₁₈	0.002	1.000	0.004	1.000	0.000	0.000
q ₁₉	0.100	1.000	0.182	0.500	0.048	0.087
q ₂₀	0.077	1.000	0.143	0.500	0.042	0.077
q ₂₁	0.077	0.500	0.133	0.500	0.043	0.080
q ₂₂	0.100	1.000	0.182	0.333	0.043	0.077
q ₂₃	0.007	1.000	0.015	1.000	0.040	0.077
q ₂₄	0.007	1.000	0.015	1.000	0.040	0.077
q ₂₅	0.007	1.000	0.015	1.000	0.040	0.077
q ₂₆	0.011	1.000	0.022	1.000	0.040	0.077
q ₂₇	0.000	1.000	0.000	1.000	0.000	0.000
q ₂₈	0.011	0.500	0.022	1.000	0.042	0.080
q ₂₉	0.006	1.000	0.011	0.667	0.083	0.148
q ₃₀	0.000	0.000	N/A	1.000	0.000	0.000

Table 6. The t-Test of two-sample assuming unequal variances to compare OpenModelica vs knowledgeMANAGER for physical models retrieval.

	OpenModelica Precision	knowledgeMANAGER Precision
Mean	0.044886824	0.851111111
Variance	0.006732369	0.068102171
Observations	30	30
Hypothesized	0	
Df	35	
t Stat	-16.14230163	
P(T<=t) one-tail	4.32626E-18	
t Critical (one tail)	1.689572458	
P(T<=t) two tail	8.65252E-18	
t Critical (two tail)	2.030107928	

**Figure 6** Precision and recall for every query and approach.

5.3 Research Limitations

Some key limitations of the presented work must be outlined. The first one relies on the sample size; our research study has been conducted in a closed world. More specifically, the physical models have been taken from a public repository and the set of queries has been automatically generated through a random walk process. That is why results in a broad or real scope could change, in terms of precision, since more complex relationships in circuits and queries could be designed. Nevertheless, the research methodology, the design of experiments and the creation of a kind of benchmark for testing retrieval processes have been demonstrated to be representative and creditable.

Regarding the generation of queries, the process creates queries similar to the way a domain expert would do. In this case, we have focused on a random combination of circuit elements due to the fact that the handmade creation of queries requires a more in-depth analysis of every circuit. This situation also implies a high probability of losing robustness due to the fact that the same domain can be interpreted according to different experts and domain discourses. However, we consider that the precision and recall metrics are helpful to make a first estimation of the advantages of using a domain ontology and knowledge representation mechanisms to retrieve physical models.

Besides, it has not been possible to fully compare both OpenModelica Connection Editor with knowledgeMANAGER because of the structure of the queries. In the text-based browser of OpenModelica it is complicated to look for several components at the same time and no advanced query mechanisms such as regular expressions are available. That is why, the precision is lower but the recall is most of times very high.

Building on the previous comments, we cannot either figure out the internal budget, methodologies, domain vocabularies, experience and background of specific domain-experts to create and query physical models. We merely observe and re-use existing public and on-line knowledge sources to provide an accurate information reuse process for physical model artifacts.

6 Conclusions and Future Work

Physical system models are not anymore isolated pieces of code to design a physical system. Current trends to develop and deploy cyber-physical systems imply the need of applying knowledge management techniques to save time and to develop safer and more secure systems. In this context, the reuse of existing and well-tested knowledge embedded into physical system models is a challenging task that can be carried out by using the proper mechanism for knowledge management. The RSHP representation model offers a flexible technique to represent any kind of knowledge through concepts and relationships. It also includes technology support through the knowledgeMANAGER tool. It seems clear that the shifting of the underlying information in physical system models to a more adequate representation improves the capabilities to discover and reuse existing knowledge.

As future work, we plan to extend the approach to any kind of physical system model (full support to the Modelica language) providing semantic engines for indexing and retrieving information. Furthermore, we will extend the experiments to make comparisons in a broad scope (tools, models and queries) releasing also the information under the principles of the OpenScience initiative.

Acknowledgements

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement N° 332830-CRYSTAL (CRITICAL sYSTEM engineering AccELeration project) and from specific national programs and/or funding authorities. This work has been supported by the Spanish Ministry of Industry.

References

- Åkesson, J., K. E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit 2010 Modeling and Optimization with Optimica and JModelica.org-Languages and Tools for Solving Large-Scale Dynamic Optimization Problems. *Computers and Chemical Engineering* 34(11): 1737–1749.
- Alvarez-Rodríguez, Jose Maria, Juan Llorens, Manuela Alejandres, and Jose Fuentes 2015 OSLC-KM: A Knowledge Management Specification for OSLC-Based Resources. In *Proceedings of the 25th Annual INCOSE International Symposium* (Accepted).
- Asgha, Syed Adeel, and Sonia Tariq 2010 Design and Implementation of a User Friendly OpenModelica Graphical Connection Editor.
- Basili, V. R., and H. D. Rombach 1991 Support for Comprehensive Reuse. *Softw. Eng. J.* 6(5): 303–316.
- Chen, Peter Pin-Shan 1976 The Entity-Relationship Model—toward a Unified View of Data. *ACM Transactions on Database Systems (TODS)* 1(1): 9–36.
- Choi, Jong-Seok, Tim McCarthy, Maneesh Yadav, et al. 2013 Application Patterns for Cyber-Physical Systems. In *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on* Pp. 52–59. IEEE.
- Davis, Randall, Howard Shrobe, and Peter Szolovits 1993 What Is a Knowledge Representation? *AI Magazine* 14(1): 17.
- Dempsey, Mike 2006 Dymola for Multi-Engineering Modelling and Simulation. 2006 IEEE Vehicle Power and Propulsion Conference, VPPC 2006.
- Desouza, Kevin C., Yukika Awazu, and Amrit Tiwana 2006 Four Dynamics for Bringing Use Back into Software Reuse. *Commun. ACM* 49(1): 96–100.
- Díaz, Irene, Juan Llorens, Gonzalo Genova, and José Miguel Fuentes 2005 Generating Domain Representations Using a Relationship Model. *Information Systems* 30(1): 1–19.
- Frakes, William, and Carol Terry 1996 Software Reuse: Metrics and Models. *ACM Computing Surveys (CSUR)* 28(2): 415–435.
- Fritzson, Peter 2015 Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach. 2. ed. New York: John Wiley & Sons Inc.
- Fritzson, Peter, and Vadim Engelson 1998 Modelica - A Unified Object-Oriented Language for System Modelling and Simulation. In *ECOOP'98 - Object-Oriented Programming, 12th European Conference, Brussels, Belgium, July 20-24, 1998, Proceedings* Pp. 67–90. <http://dx.doi.org/10.1007/BFb0054087>.
- Groza, Tudor, Siegfried Handschuh, Tim Clark, S Buckingham Shum, and Anita de Waard 2009a A Short Survey of Discourse Representation Models.
- Groza, Tudor, Siegfried Handschuh, Tim Clark, S Buckingham Shum, and Anita de Waard 2009b A Short Survey of Discourse Representation Models.
- Guo, Jiang, and others 2000 A Survey of Software Reuse Repositories. In *Engineering of Computer-Based Systems, IEEE International Conference on the* Pp. 92–92. IEEE Computer Society.
- Hayes, Patrick 2004 RDF Semantics. World Wide Web Consortium. <http://www.w3.org/TR/rdf-mt/>.
- Hull, Richard, and Roger King 1987 Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys (CSUR)* 19(3): 201–260.
- Jacobson, Ivar, Martin Griss, and Patrik Jonsson 1997 Software Reuse: Architecture, Process and Organization for Business Success. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.
- Karlsson, Even-André, ed. 1995 Software Reuse: A Holistic Approach. New York, NY, USA: John Wiley & Sons, Inc.
- Kim, Kyoung-Dae, and Panganamala R Kumar 2012 Cyber-physical Systems: A Perspective at the Centennial. *Proceedings of the IEEE 100(Special Centennial Issue):* 1287–1308.
- Kim, Minyoung, M-O Stehr, Jinwoo Kim, and Soonhoi Ha 2010 An Application Framework for Loosely Coupled Networked Cyber-Physical Systems. In *Embedded and*

- Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on Pp. 144–153. IEEE.
- Land, Rikard, Daniel Sundmark, Frank Lüders, Iva Krasteva, and Adnan Causevic 2009 Reuse with Software Components—a Survey of Industrial State of Practice. In *Formal Foundations of Reuse and Domain Engineering* Pp. 150–159. Springer.
- Llorens, Juan, Jorge Morato, and Gonzalo Genova 2004 RSHP: An Information Representation Model Based on Relationships. In *Soft Computing in Software Engineering*. Ernesto Damiani, Mauro Madravio, and LakhmiC. Jain, eds. Pp. 221–253. Studies in Fuzziness and Soft Computing. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-540-44405-3_8.
- Martin-Villalba, Carla, Alfonso Urquia, and Sebastian Dormido 2008 An Approach to Virtual-Lab Implementation Using Modelica. *Mathematical and Computer Modelling of Dynamical Systems* 14(4): 341–360.
- Mcilroy, Doug 1969 Mass-Produced Software Components. In *Proceedings of Software Engineering Concepts and Techniques*. J. M. Buxton, P. Naur, and B. Randell, eds. Pp. 138–155. Garmisch, Germany: NATO Science Committee. <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF>.
- Mili, Ali, Rym Mili, and Roland T Mittermeir 1998 A Survey of Software Reuse Libraries. *Annals of Software Engineering* 5: 349–414.
- Mili, Hafedh 2002 *Reuse Based Software Engineering: Techniques, Organization and Measurement*. New York: Wiley.
- Morisio, M., M. Ezran, and C. Tully 2002 Success and Failure Factors in Software Reuse. *IEEE Transactions on Software Engineering* 28(4): 340–357.
- Nonaka, Ikujiro, and Hirotaka Takeuchi 1995 *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. New York: Oxford University Press.
- Otter, Martin, Torsten Blochwitz, and Martin Arnold 2013 Functional Mock-up Interface for Model Exchange and Co-Simulation: 1–120.
- Rajkumar, Ragunathan Raj, Insup Lee, Lui Sha, and John Stankovic 2010 Cyber-Physical Systems: The next Computing Revolution. In *Proceedings of the 47th Design Automation Conference* Pp. 731–736. ACM.
- Ryman, Arthur G., Arnaud Le Hors, and Steve Speicher 2013 OSLC Resource Shape: A Language for Defining Constraints on Linked Data. In LDOW.
- Samlaus, Roland, and Peter Fritzson 2015 Semantic Validation of Physical Models Using Role Models. *Simulation* 91(4): 383–399.
- Schamai, Wladimir, Peter Fritzson, and Christiaan J. J. Paredis 2013 Translation of UML State Machines to Modelica: Handling Semantic Issues. *Simulation* 89(4): 498–512.
- Smolárová, Mária, and Pavol Návrát 1997 Software Reuse: Principles, Patterns, Prospects. *CIT. Journal of Computing and Information Technology* 5(1): 33–49.
- The Reuse Company Inc. 2014 knowledgeMANAGER (KM). Industry website. knowledgeMANAGER. <http://www.reusecompany.com/knowledgemanager>, accessed October 15, 2014.
- Thüm, Thomas, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake 2014 A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Computing Surveys* 47(1): 1–45.
- Tracz, Will 1995 *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Valášek, M, P Steinbauer, J Kolář, and J Dvořák 2003 Concurrent Design of Railway Vehicles by Simulation Model Reuse 43(6): 9–15.
- Wellstead, Peter E 1979 *Introduction to Physical System Modelling*. London: Academic Press.
- Winsberg, Eric 2001 *Simulations, Models, and Theories: Complex Physical Systems and Their Representations*. *Philosophy of Science* 68(S1): S442.

Mass Conserving Models of Vapor Compression Cycles

Christopher R. Laughman Hongtao Qiao

Mitsubishi Electric Research Laboratories {laughman,qiao}@merl.com

Abstract

Many dynamic models of vapor compression systems experience nonphysical variations in the total refrigerant mass contained in the system when common modeling approaches are used. Rather than use the traditional state variables of pressure and specific enthalpy, the use of density as a state variable can eliminate these variations. A set of test models is developed in Modelica to study the effect of the state variable selection on the overall system charge, and results indicate that this alternative approach has significant benefits for maintaining a specified mass of refrigerant in the cycle.

Keywords: vapor compression cycle, simulation, mass conservation

1 Introduction

Trends toward increased integration in building and transportation systems, as well as perennial demands for improved system performance, have continued to encourage interest in the development of dynamic models of vapor compression cycles. Such dynamic cycle models can be used for a variety of purposes, including system design, specification, control, and fault diagnostics, and can be applied to a wide variety of residential, commercial and industrial applications to understand and predict the behavior of field-installed systems. These dynamic models can also be coupled with other systems to examine and design the behavior of systems-of-systems to achieve specified requirements for the overall system and satisfy constraints that must be enforced on the physical hardware.

This wealth of interest in dynamic models of vapor compression cycles has resulted in a corresponding growth in both the literature and the number of documented models for these cycles (Li et al., 2014b). The Modelica language is particularly appropriate for the development of these system models, due to its object-oriented, declarative, and acausal modeling approach. This can be seen in the variety of references that have been published over the past 15 years regarding models of vapor compression cycles, such as those found in Li et al. (2014a), among many others.

The performance of physical system models is often evaluated by comparing particular characteristics or outputs of their simulations to the related characteristics of an experimentally observed system. Since, as George E.P. Box said, “all models are wrong, but some are useful,” (Box and Draper, 1987), model creators and users must examine the most salient characteristics of their model to ensure that it accurately describes the behavior of interest. This is particularly important for such complex physical systems as vapor compression cycles; it is essential that engineers compare and validate dynamic cycle models against known experimental behavior and data before expecting to obtain reliable model output. One such variable that can easily be compared between simulation and experiment is the the cycle’s refrigerant mass inventory, or charge, which is usually known to a fairly high degree of precision, and is also constant over extended time intervals. Such stability and ease of measurement is theoretically well-suited to use in model parameterization and calibration, and is convenient for study in dynamic system models.

Unfortunately, many model formulations for vapor compression cycles demonstrate significant variations in the total system charge (Cecchinato and Mancini, 2012) that do not correspond to observed behavior in experimental systems. This is significant for a few reasons; perhaps the most important of these is that the dynamics associated with the variations in the cycle charge will be coupled to the other system dynamics and introduce aberrant behavior that would not be observed in an experimental system. In addition, the dynamics of the refrigerant mass may also be important of themselves, particularly as pertains to ongoing efforts to develop cycles with minimized refrigerant charge (Corberan et al., 2011). Finally, the relative ease and precision with which the refrigerant mass can be measured, particularly in relation to other quantities such as the specific enthalpy, can be invaluable in calibrating dynamic models of these systems to experimental data.

One contribution to the related field of evaporator charge management was made by Cecchinato and Mancini (2012), in which the authors develop a moving-boundary formulation of a single evaporator that conserves refrigerant mass. Previous work related to the dynamics associated with the cycle charge also includes that of Bonilla et al. (2012), in which the authors study

the effect of system oscillations and numerical instability resulting from variations in the density in an evaporator. Other work with a similar focus includes that of Tummescheit (2002), which discusses both chattering (oscillations around a phase boundary) and the selection of different state variables due to different parameterizations for the equations of state for various fluids.

There are two primary objectives of this paper: exploring the causes of the variations in the cycle charge, and developing an alternative modeling approach that conserves refrigerant mass. This study will be done via the use of a simplified cycle model, developed in Modelica, that eliminates extraneous complexity yet maintains the salient characteristics of models that cause variations in the cycle charge. While common cycle models have many important additional characteristics, such as the use of detailed heat transfer or frictional pressure drop correlations, these characteristics are not essential to the analysis of, or solution to, the variations in the cycle charge. One additional effect that is significant for experimental systems but has been neglected for this initial study is that of the refrigerant oil; while some of the refrigerant charge in experimental systems is inevitably dissolved in the oil and a charge inventory that ignores this effect will inevitably be lower than experimentally observed system charge, the challenges inherent in modeling the refrigerant-oil interactions and the need for initial work in this area elicited a focus on single-component working fluids.

Following this introduction, Section 2 discusses the causes of the variation in the cycle charge in the context of the finite volume pipe model, as well as a method of eliminating these variations. Section 3 presents a discussion of the construction and implementation of the component models used in the simplified cycle models which are both conservative and nonconservative, as well as an approach for initializing these models to achieve a specified cycle charge. The results of simulating these modified models to eliminate the fluctuations in cycle charge are discussed in Section 4, while the final section summarizes the work presented in the paper and suggests fertile areas for exploration future work.

2 Cycle Mass Variation

Basic vapor compression cycles consist of a compressor, an expansion valve, and two heat exchangers. Common simulation architectures are designed to take advantage of the different timescales for the dynamics of the different components; since the time constants of the compressor and expansion valve are such smaller than those of the heat exchangers, algebraic models are used for these components, and dynamic models are used for the heat exchangers. One common type of models for the heat exchanger dynamics used in this research are so-called finite volume models, which use the method of lines to

discretize the partial differential equations (PDEs) describing the mass, momentum, and energy conservation in the system. The resulting model formulation consists of a set of ordinary differential equations (ODEs) that can be integrated forward in time to study the dynamics of the system, as well as a set of algebraic constraints including those introduced by the compressor and expansion valve models. While the high complexity of the finite volume models makes them somewhat slower than other heat exchanger modeling approaches, their ability to describe spatial variations in the heat exchanger behavior has made them quite popular (Elmqvist et al., 2003; Franke et al., 2009; Laughman, 2014).

As is the case with the development of any physical system model, it is essential to clearly define the purpose for which a model is constructed to ensure that it uses an appropriate set of assumptions to describe the desired behavior. Since the behavior of the refrigerant mass in the cycle are the focus of this research, the models constructed in this paper only describe the behavior of the working fluid in the pipe, rather than the dynamics of the coupled primary fluid / tube wall / secondary fluid system of a prototypical air-source vapor compression cycle. The conservation equations were also simplified by neglecting both gravitational forces and axial heat conduction in the direction of the fluid flow. Other model assumptions used in this work include that of one-dimensional pipe flow, thermodynamic equilibrium in each discrete volume of the refrigerant pipe at each instant in time, and a homogeneous flow field in the two-phase region, meaning that the liquid and vapor velocities are equal. These assumptions were employed to avoid additional complexity in the models in an effort to focus on the underlying causes of variations in the cycle mass.

Under these assumptions, the PDEs describing the conservation equations for a volume of fluid in the refrigerant pipe are

$$\frac{\partial(\rho A)}{\partial t} + \frac{\partial(\rho A v)}{\partial x} = 0 \quad (1)$$

$$\frac{\partial(\rho v A)}{\partial t} + \frac{\partial(\rho v^2 A)}{\partial x} = -A \frac{\partial P}{\partial x} - F_f \quad (2)$$

$$\frac{\partial(\rho u A)}{\partial t} + \frac{\partial(\rho v h A)}{\partial x} = v A \frac{\partial P}{\partial x} + v F_f + \frac{\partial Q}{\partial x}, \quad (3)$$

where additional information about the symbols and nomenclature used in these equations can be found in the table at the end of this paper. The Reynolds transport theorem can be used to relate the changes in state for a control volume of fixed dimension to the the fluid flowing into and out of that control volume. The resulting expressions can then be discretized to generate a set of

ODEs, e.g.,

$$\frac{d(\rho_j V_j)}{dt} = \dot{m}_k - \dot{m}_{k+1} \quad (4)$$

$$\frac{d(\dot{m}_i l)}{dt} = \rho_j v_j^2 A_j - \rho_{j+1} v_{j+1}^2 A_{j+1} + \frac{A_j + A_{j+1}}{2} (P_{j+1} - P_j) + F_{f,i} \quad (5)$$

$$\frac{d(\rho_j u_j A_j)}{dt} = \dot{H}_k - \dot{H}_{k+1} + v_j A_j (P_{j+1} - P_j) + v F_{f,i} + \dot{Q}_j, \quad (6)$$

where the set of ODEs corresponds to the number of volumes used to subdivide the length of the refrigerant pipe, and the indices refer the fact that we are using a staggered flow grid (Patankar, 1980). In these equations, the i indices are referred to the momentum grid, the j indices are referred to the thermal grid, and the $k = j + 1$ indices refer to the boundaries of the thermal grid. In addition, the term \dot{H}_k is defined as

$$\dot{H}_k = \dot{m}_k \bar{h}_{upstream,j}, \quad (7)$$

and the mixed-cup specific enthalpy \bar{h} is equal to the *in situ* specific enthalpy under the homogeneous flow assumption (Laughman, 2014).

Thermodynamic property relations also play an important role which is complementary to the differential equations of fluid motion. These property relations, which are also algebraic, describe the relations between the intensive and extensive fluid properties for a given volume of fluid in thermodynamic equilibrium. These properties include temperature, pressure, specific enthalpy, and density, among many others. As a result of the Gibbs phase rule, there are two degrees of freedom for a single-component pure fluid when there is only one phase present, so that knowledge of two intensive properties is sufficient to determine any other property. When there are two-phase flows, there is only one degree of freedom, but the specification of an intensive mixture property is also needed to determine the state of the two-phase mixture (Bejan, 2006). For example, the specification of pressure P and mixture specific enthalpy h will theoretically allow the calculation of any other properties in the thermodynamic phase space.

The calculation of thermophysical properties for dynamic simulation generally needs to be very fast and accurate, due to the number of function evaluations used in a typical system model. As a result, the use of standard equations of state is discouraged in favor of other interpolating methods, such as cubic polynomials or splines (Aute and Radermacher, 2014). Such methods use function approximation to describe each of a set of desired properties as a function of a much more limited set of properties that are calculated at each time step in the simulation. Many thermophysical property routines

for refrigerants use P and h as coordinates in the function approximation space to quickly calculate the variety of necessary properties.

The construction of a dynamic model of a refrigerant pipe must take into consideration both the structure of the equations of fluid motion, as well as the implementation of the thermophysical property calculation methods, to generate a computationally efficient simulation. The selection of an infelicitous set of coordinates in which to integrate the conservation equations 4-6 can result in the generation of a large set of nonlinear equations that must be solved to calculate the fluid properties at every time step and for every fluid volume, resulting in potential numerical and practical challenges.

The most common approach taken in this regard is the selection of pressure P and specific enthalpy h as the state variables for the equations of motion, since these are often also used as the coordinates for calculating the fluid properties. The derivatives of $M(P, h)$ and $U(P, h)$ in the above equations can thus be written as

$$\frac{dM}{dt} = V \left(\frac{d\rho(P, h)}{dt} \right) \quad (8)$$

$$= V \left(\left. \frac{\partial \rho}{\partial P} \right|_h \frac{dP}{dt} + \left. \frac{\partial \rho}{\partial h} \right|_P \frac{dh}{dt} \right) \quad (9)$$

$$\frac{dU}{dt} = V \left(\frac{d(\rho(P, h)u(P, h))}{dt} \right) \quad (10)$$

$$= V \left[\left(h \left. \frac{\partial \rho}{\partial P} \right|_h - 1 \right) \frac{dP}{dt} + \left(\left. \frac{\partial \rho}{\partial h} \right|_P h + \rho \right) \frac{dh}{dt} \right]. \quad (11)$$

The use of these property relations, along with the `stateSelect` attribute, can help the Modelica compiler to select P and h as the state variables for the model. By selecting these properties as state variables, they can be integrated by the solver used in a given Modelica tool, such as DASSL or Radau IIa.

The selection of a set of coordinates for the system can have a significant impact on many other variables of the system. One particular variable that is strongly affected by this choice of state variables is the total mass of the system M_{total} . Since no mass is stored in the compressor or expansion valve models, an expression for M_{total} can be developed by summing all of the masses for the individual control volumes in the pipe model, e.g.,

$$M_{total} = \sum_k \rho_k V_k = \sum_k \rho_k(P, h) V_k. \quad (12)$$

Because the integration of the state variables results in some error, however, it is important to note that a more accurate description of this sum might be

$$M_{total} = \sum_k \hat{\rho}_k(P + \varepsilon, h + \varepsilon) V_k, \quad (13)$$

where ε is the error tolerance of the integration routine and $\hat{\rho}$ represents the numerical approximation of

ρ . While these integration errors are not problematic in many fluids for which the relation between P , h , and ρ is nearly linear, two-phase refrigerant flows experience large changes in density as the fluid passes from the liquid region into the two-phase region. These large density derivatives can effectively amplify small deviations in either P or h , resulting in large changes in the density between subsequent time steps. Consequently, small errors in the integration of both of these quantities can accumulate quickly and lead to significant and unexpected changes in the total system mass.

Further consideration of Equation 12 suggests an alternative choice of state variables that can reduce these undesirable changes in the refrigerant mass; since the ultimate objective of reducing nonphysical variations in the system charge is equivalent to reducing the errors in the cell density calculations, the selection of ρ as a state variable will allow the integrator to minimize the errors in the density directly, rather than through $\rho(P, h)$. While this choice may appear to be unconventional because of the potential for numerical chattering caused by the large density changes that accompany the movement of the fluid state across the saturated liquid line, the choice of P and ρ as state variables will eliminate the amplification of errors in the density calculation, resulting in a corresponding reduction in the variation of the total system mass.

The alternative formulation of the state variables results in the following expressions for the derivatives of $M(P, \rho)$ and $U(P, \rho)$ for each control volume, e.g.,

$$\frac{dM}{dt} = V \frac{d\rho}{dt} \quad (14)$$

$$\frac{dU}{dt} = \frac{d(\rho u(P, \rho)V)}{dt} \quad (15)$$

$$= V \left[\left(\rho \frac{\partial h}{\partial P} \Big|_{\rho} - 1 \right) \frac{dP}{dt} + \left(\rho \frac{\partial h}{\partial \rho} \Big|_P + h \right) \frac{d\rho}{dt} \right]. \quad (16)$$

As might be expected, the selection of ρ does also impose additional costs to the simulation. Perhaps the most significant of these is that the use of ρ as a state variable will result in smaller time steps because of the large values of the derivatives at low static qualities of the flow. In addition, the selection of these state variables will also have an effect on the final set of equations that are generated because the change in coordinates will result in the construction of a different set of equations to calculate the remaining fluid properties, such as the calculation of $h(P, \rho)$. In the case that these equations are nonlinear, the simulation time could also be longer than would be for the case with the selection of the original state variables. However, these costs may be outweighed by the benefit of having a constant cycle charge.

Another alternative method for describing the dynamics of the differential control volume involves expanding the number of state variables to include pressure, specific

enthalpy, and density. While this approach does result in a larger number of state variables, it has the advantage of simultaneously minimizing the variations in system charge while enabling the use of P and h for calculating other refrigerant properties. Such a method uses the same differential equations as the (P, ρ) model, but also includes the additional ODE

$$\frac{dh}{dt} = \frac{\partial h}{\partial P} \Big|_{\rho} \frac{dP}{dt} + \frac{\partial h}{\partial \rho} \Big|_P \frac{d\rho}{dt}. \quad (17)$$

It is also important to note that the set of property derivatives $\partial h / \partial P$ and $\partial h / \partial \rho$ from Equations 14 and 16 do not need to be separately calculated in the property routine to use P and ρ as state variables. The original set of property derivatives can instead be manipulated to provide the needed derivatives, i.e.,

$$\frac{\partial h}{\partial P} \Big|_{\rho} = - \frac{\partial \rho}{\partial P} \Big|_h \frac{\partial h}{\partial \rho} \Big|_P \quad (18)$$

$$\frac{\partial h}{\partial \rho} \Big|_P = \frac{1}{\frac{\partial \rho}{\partial h} \Big|_P}. \quad (19)$$

3 Mass Conserving Models

A simplified cycle model, described in the following section, was developed to evaluate the efficacy of these different approaches at maintaining a specified cycle charge. Details about the components and construction of this test cycle model will be discussed in this section, as well as the means of initializing this cycle to achieve a specified system charge.

3.1 Component Models

The simplified cycle model developed in this section includes three components: a refrigerant pipe, a pump, and an “enthalpy adjuster”. These components were used to create a system cycle model which maintained mass and energy balances. While the main focus of this work is the refrigerant pipe, the pump is needed to define a relation between the mass flow rate and the pressure drop, so that these variables can be controlled and varied to examine their effect on the total cycle mass. An additional component, referred to as an enthalpy adjuster, was also used to enforce the conservation of energy throughout the system; this component included no pressure drop, but only modified the enthalpy of the working fluid flowing through it so that energy was conserved over the cycle. Neither the pump nor the enthalpy adjuster stored any refrigerant mass; consequently, these components had no state and imposed only algebraic constraints on the system to achieve a desired system balance point. The state variables were therefore only associated with the refrigerant pipe.

A simplified pipe model, governed by the equations described in Section 2, was developed to test the impact of the state variable selection on the dynamics of the cycle charge. In addition to the governing ODEs, these models also required the inclusion of a set of closure relations describing the heat transfer and the frictional pressure drop. An ideal heat transfer connection was assumed for the sake of simplicity, so that the thermal energy was directly added to the refrigerant stream in each control volume, rather than being governed by the temperature gradients between the refrigerant pipe wall and the bulk fluid. A simplified momentum equation that only accounts for the steady-state frictional pressure drop in both the single and two-phase regions was also assumed, in which

$$\Delta P = K \frac{\Delta P_0}{\dot{m}_0^2} \dot{m}^2, \quad (20)$$

and the nominal values of ΔP_0 , \dot{m}_0 , and the adjustable constant of proportionality K were set at the top level of the model. A numerical regularization method was also implemented to improve the numerical robustness of the model for small values of the mass flow rate and pressure drop (Casella et al., 2006).

One feature of the pipe model that was particularly important to this work was the ability to use different models for the relations between the property differentials. This was achieved by implementing the set of differential models as a replaceable model inside the larger pipe model. Each pipe model includes its own differential volume model, but computes the same terms dMs and dUs . While each of these underlying differential models implements different relationships between the properties, the instantiating pipe model only needs to equate the differentials of the mass and internal energy to the terms on the right hand side of Equations 4-6. This is demonstrated in the following simplified excerpt from the refrigerant pipe model.

```
// DIFFERENTIAL VOLUME MODEL
replaceable model DifferentialModel =
  DifferentialModel_ph
  constrainedby
  PartialDifferentialModel;

DifferentialModel diffVolume(
  redeclare Medium=Medium,
  n=n,
  fluidVolumes=fluidVolumes,
  ps={mediums[k].p for k in 1:n},
  hs={mediums[k].h for k in 1:n},
  ddhps={mediums[k].ddhp for k in 1:n},
  ddphs={mediums[k].ddph for k in 1:n},
  stateChoice=stateChoice);

equation
  dms = diffVolume.dms;
  dUs = diffVolume.dUs;
```

By further establishing a `PartialDifferential-Model` from which all of these differential models can inherit, the differential volume model can be replaced while maintaining some moderate restrictions on the possible types of replacement, enabling the state variables to be changed without changing any of the other equations in the pipe model.

This implementation of these differential volume models also required the careful use of the `stateSelect` attribute, as the selection of states was based upon the choice of state variables managed with the differential volume model. A `ThermoStates` enumeration with literals including `states_ph`, `states_pd`, and `states_phd` was therefore used to coordinate the use of a given differential volume model and the corresponding state selection attribute for the Modelica compiler.

A pump model and an enthalpy adjuster model were also created to study the closed loop cycle dynamics. The pump model used a scaled version of the basic relationship between mass flow rate and pressure drop (Equation 20) to calculate the pressure rise across the pump for the nominal pump speed that is inversely proportional to the pressure drop for pipe model including a given number of control volumes, e.g.,

$$\dot{m} = \left(\frac{N}{N_{nom}} \right) \frac{\dot{m}_0 \sqrt{\Delta P_0}}{\sqrt{\Delta P \left(\frac{N_{nom}}{N} \right)^2}}. \quad (21)$$

As no mass was stored in this component, the mass flow rates into and out of the pump were equal, and the energy change across the pump was a quadratic function that compensated for the change in enthalpy across the pipe due to pressure loss. This term was much smaller than the energy change in the pipe due to the heat flux into the pipe. As was the case for the pressure drop model of the pipe, regularization methods were also used to compensate for numerical singularities.

An analogous enthalpy adjuster model was also created to compensate for the change in the specific enthalpy across the pipe due to the applied heat flux. This model included no pressure drop or mass storage, and only modified the specific enthalpy for the working fluid travelling to include the effect of the total applied thermal energy gain as the fluid travels through the pipe. Consequently, the equations describing the simplified model used to fulfill the energy balance for the overall system are

$$\dot{m}_{out} = \dot{m}_{in} \quad (22)$$

$$P_{out} = P_{in} \quad (23)$$

$$h_{out} = \text{inStream}(h_{in}) + \dot{Q}_{in}/\dot{m}_{in} \quad (24)$$

$$h_{in} = \text{inStream}(h_{out}) - \dot{Q}_{in}/\dot{m}_{in}. \quad (25)$$

This model is very similar to that of `Modelica.Fluid.Pipes.StaticPipe`, but also

includes a change in the outlet enthalpy corresponding to the applied heat gain. Stream connectors and regularization methods around zero mass flow rate were also used in these individual components to improve the numerical robustness of the simulation.

3.2 Initialization

The problem of achieving a specified constant charge for a cycle simulation can be effectively split into two related problems: the initialization of the simulation so that the cycle mass starts at the specified value, and the maintenance of the cycle charge at that value over the duration of the simulation. While the previous sections of the paper address how to maintain the cycle charge at a constant value, this brief section addresses the means by which a specific value of the cycle charge may be attained. In general, the total refrigerant mass contained in the cycle at initialization depends on the initial refrigerant state in each volume of the system. Because the refrigerant state at zero mass flow rate is relatively easier to determine, the system was initialized as this condition so that the pump speed was zero and there was no heat flux applied to the pipe or the enthalpy adjuster, and then these inputs were turned on after the conclusion of the initialization transient.

The initial conditions for the system were developed using basic thermodynamic reasoning. The specification of a value of cycle charge M_{total} for a given system volume V effectively specifies the average density of the fluid in the system ρ_{init} ; this specifies one variable that determines the state of the system. Independent specification of one other variable for the system, such as the system pressure P_{init} at zero pump speed and zero heat flux, determines the state of the refrigerant in the system. The specific enthalpy h_{init} for every component and control volume can therefore be directly calculated from this refrigerant state in a set of initial equations. Since it is common to initialize most components with pressure and specific enthalpy, these calculated initial values for the pressure and specific enthalpy of the working fluid were then used to initialize all of the components in the system to achieve the desired cycle charge.

4 Results

The models described in Section 3 were implemented in Modelica and tested to evaluate the efficacy of the proposed strategy for maintaining a constant cycle mass. Three related models were created with identical geometric parameters and input waveforms. These models used the R410a refrigerant property model included in the AirConditioning/ThermoFluidPro library, written by Modelon (Modelon AB, 2015), as well as the simple relationship between frictional pressure drop and mass flow rate described in Equation 20, where $\Delta P_0 = 500$ Pa

and $\dot{m}_0 = 10$ g/s. Other salient parameters of the model are included in Table 1. These models were tested in sim-

Table 1. Common parameters for the test cycle models.

Parameter Name	Value
Pipe diameter	8 mm
Pipe length	12 m
Maximum heat input	130 W/cell (3120 W total)
Initial pressure	1 MPa
Initial system charge	150 g
Number of pipe control volumes	24

ulation using Dymola 2015 FD01, and were executed on an i7 PC with 8G of RAM.

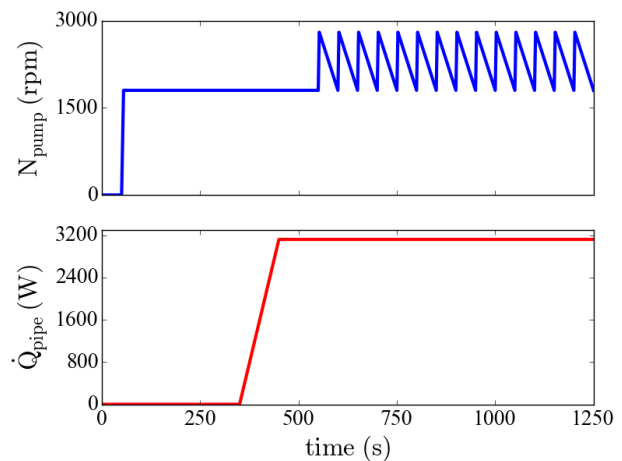


Figure 1. Inputs of pump speed (upper) and heat input (lower) applied to the test cycle.

Because the variations in the cycle charge are related to phase transitions in the fluid volumes across the liquid saturation line, a series of inputs was designed to repeatedly produce these transitions in an effort to induce variations in the cycle charge. These input waveforms, both for the pump speed and the heat source, are illustrated in Figure 1. After the cycle was initialized with the specified refrigerant mass and zero mass flow rate, the pump speed was initially ramped up at 50 seconds from 0 to 1800 rpm over 5 seconds. The resulting transients were then allowed to subside before ramping up the heat source at 350 seconds from 0 to 3120 W over 100 seconds, with the heat being distributed equally over each of the 24 control volumes in the pipe. Finally, a ramp sawtooth waveform was applied to the pump speed to repeatedly cause transitions across the liquid saturation line; the resulting pump speed had a minimum value of 1800 rpm, a maximum value of 2800 rpm, a period of 50 seconds, and a duty ratio of 0.052. All of the simulations used identical input waveforms, and were integrated by using the DASSL solver.

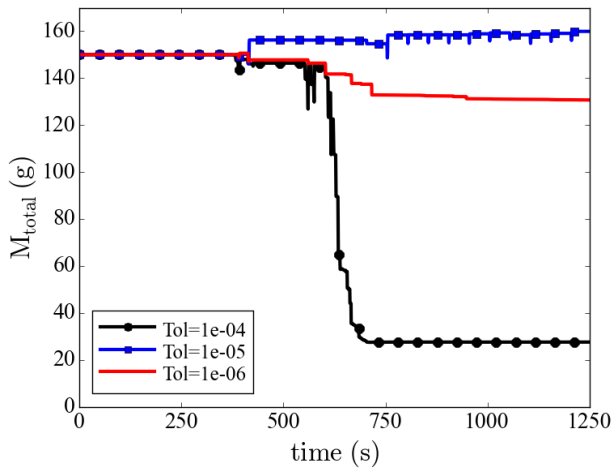


Figure 2. Total cycle mass for three different numerical tolerances with identical applied inputs.

The effect of this waveform on the model using P and h as state variables are illustrated in Figure 2. While many notable features are evident, perhaps the most striking is the amount of variability in the total cycle charge. Such large changes in the total cycle charge can be quite problematic, as they will have a significant impact on the behavior of the cycle. The amount of variation in the total cycle mass is strongly correlated with the tolerance of the solver, suggesting that it is indeed related to the integration tolerances. Moreover, the changes in the mass inventory usually occur by steps, suggesting the presence of a discontinuity that gives rise to these changes.

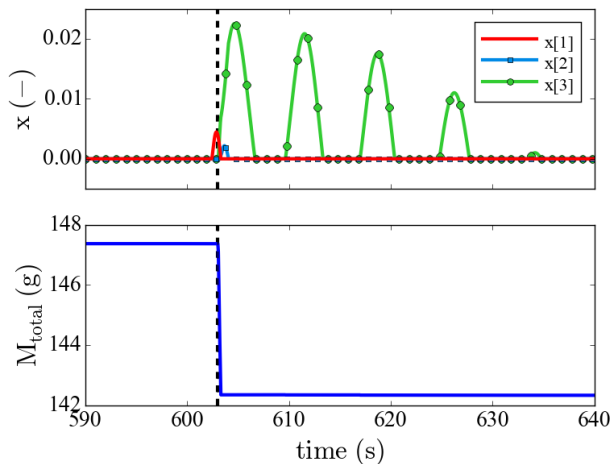


Figure 3. Static quality x at the first, second, and third control volumes in the pipe during the increasing portion of the pump speed waveform, as well as the total system charge at the same moment.

Figure 3 illustrates the relation between the discontinuity caused by the changes in the static quality $x = M_{vap}/M_{total}$ for control volumes 1, 2, and 3 and the variations in the total system charge. The dashed line drawn at $t = 603$ seconds shows a strong correlation between the

time that the static quality for all three of these control volumes goes above zero and the time of the step discontinuity in the total system charge. It is also particularly interesting to note that while the quality of the third control volume increases above zero a number of subsequent times in this plot, there are no other variations in the total system charge. This phenomenon suggests that the variations in the refrigerant charge are related not only to a transition across the liquid saturation line, but also to the rate and duration of this transition. The small magnitude of the abrupt excursions over $x = 0$ for control volumes 1 and 2 which are associated with large changes in the refrigerant density, as well as the corresponding large changes in the cycle mass, is compatible with the assertion that the variations in the total system charge could be caused by the errors in the state variables.

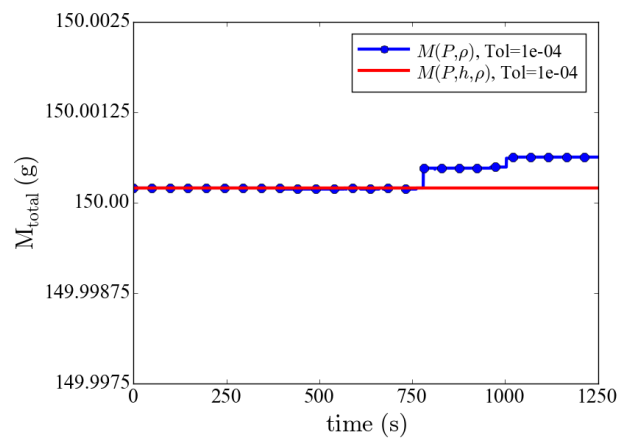


Figure 4. Cycle mass inventory for $M(P, \rho)$ and $M(P, h, \rho)$ models, with an integration tolerance of $1e-04$.

In comparison to the large variations in the total system charge exhibited in Figure 2 for the system using (P, h) as state variables, the minuscule variations present in Figure 4 demonstrate that the models that use either (P, ρ) and (P, h, ρ) as state variables have much improved behavior. The variations in the mass for both of these cycles are on the order of 0.25 milligrams, or $1.7 \times 10^{-4}\%$ of the total cycle charge. This compares quite favorably to the output of the simulation of the (P, h) model with the same tolerance, which resulted in an 82% change in the total cycle charge. Further reductions in the error tolerance for the (P, ρ) and (P, h, ρ) simulations will result in a corresponding reduction in the variation in the total cycle charge.

Additional insights can be gained from the information contained in Table 2, which compares the errors in the simulations and the total time required to run each simulation for different sets of state variables and error tolerances. The errors in this table were generated by calculating the maximum deviation between the total system charge and 150.0 grams, which was the specified charge. As might be expected, the error in the total system charge is far greater for the model with the (P, h)

Table 2. Max and percentage errors and CPU time for different choices of state variables and integrator tolerances.

State Var	Tol	Max Error	% Error	Time
$M(P,h)$	1e-4	-122.6 g	81.7%	277 s
	1e-5	3.96 g	2.6%	127 s
	1e-6	-19.3 g	12.8%	1925 s
$M(P,\rho)$	1e-4	1.9e-4 g	1.2e-4%	766 s
	1e-5	2.0e-4 g	1.3e-4%	1250 s
	1e-6	2.0e-4 g	1.3e-4%	1374 s
$M(P,h,\rho)$	1e-4	2.0e-4 g	1.3e-4%	137 s
	1e-5	2.0e-4 g	1.3e-4%	315 s
	1e-6	2.0e-4 g	1.3e-4%	450 s

state variables than for the other models. One particularly striking and counterintuitive trend is the decrease in the simulation time for the (P,h) models that accompanies the reduction in the tolerance from 1e-4 to 1e-5; this can be attributed to the stiffness of the system of equations during the abrupt changes in the mass inventory in the simulation with the higher tolerance. It is also interesting to note that the simulation time for the (P,h) model with a tolerance of 1e-6 is much greater than for any of the other simulations for any combination of state variables. This can potentially be attributed to the presence of so many discontinuities in the simulation waveform due to the changes in the refrigerant mass; since the solver must take very small time steps past each discontinuity to maintain the specified error tolerance, the sum effect of these discontinuities is that the average time step of the solver must be much smaller than might otherwise be necessary.

Comparison of the simulation time of the (P,h) models to the (P,ρ) models indicates that the (P,ρ) models are slower, as expected, because the large variations in refrigerant density cause the solver to take correspondingly smaller time steps. Finally, it is also evident from Table 2 that the (P,ρ) and (P,h,ρ) methods have identical accuracy for practical intents and purposes, but the time required to run the (P,h,ρ) simulations is much smaller than that of the (P,ρ) simulations. This can potentially be attributed to the nonlinear equations that must be solved to calculate $h(P,\rho)$ when h is not used as a state variable.

5 Conclusions and Further Work

Over the course of this paper, the causes of variations in the total system charge were studied and two alternative selections of the state variables that can essentially eliminate such variations were proposed. The effect of these different state variable selections was demonstrated on a simplified cycle model, and the manifestations of the underlying causes for the cycle variation when P and h are solely used as state variables were examined by

analyzing the simulation output. While both the (P,ρ) and (P,h,ρ) models had similar accuracy for simulating the total system charge, the (P,h,ρ) models simulated much faster because $h(P,\rho)$ does not have to be calculated when it is also included as a state variable. Moreover, though one ostensible motivation for using (P,h) as state variables is the speed by which the property calculations can be executed, the dynamics associated with the variation in total system charge can somewhat ironically result in simulations that take longer to run than simulations with (P,ρ) as state variables because of the small step sizes required. Models for refrigerant pipes that include either (P,ρ) or (P,h,ρ) as state variables could therefore result in simulations that are both faster and more accurate than might be possible with a choice of (P,h) as state variables.

The results obtained in this work may be extended in a number of directions for future investigation. As suggested in the introduction, an extension of these methods to models which describe the behavior of refrigerant/oil mixtures would be quite valuable. In addition, an error analysis to rigorously demonstrate the causes of these cycle variations would clarify the observations discussed in this paper, and a study of the energy conservation for the system might also provide interesting results. While it is expected that these general trends would hold for different solvers, choices of the nominal attributes of the states, or reference values of the specific enthalpy, further work to explore such trends would be beneficial. Additional study of alternate thermodynamic coordinates might also yield fruitful results; for example, specific entropy is sometimes used to decouple the hydraulic and thermal equations describing fluid flow, and the selection of this or alternate coordinates may also be relevant to these applications. We hope that future studies of these and associated phenomena will continue to yield new insights into these complex and fascinating systems.

Nomenclature

A	cross-sectional area
F_f	frictional pressure drop
H	enthalpy flow rate
K	proportionality constant for $\dot{m} \rightarrow \Delta P$ relation
M	mass
N	pump speed
P	pressure
\dot{Q}	heat transfer rate
U	internal energy
V	volume
h	<i>in situ</i> specific enthalpy
\bar{h}	“mixed-cup” specific enthalpy
\dot{m}	mass flow rate
t	time
u	specific internal energy

v velocity
 ρ density
 $\hat{\rho}$ numerical approximation of density

S.V. Patankar. *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publishing Co., 1980.

H. Tummescheit. *Design and Implementation of Object-Oriented Model Libraries using Modelica*. PhD thesis, Lund Institute of Technology, 2002.

References

- V. Aute and R. Radermacher. Standardized polynomials for fast evaluation of refrigerant thermophysical properties. In *International Refrigeration and Air-Conditioning Conference at Purdue*, 2014.
- A. Bejan. *Advanced Engineering Thermodynamics*. Wiley, 3 edition, 2006.
- J. Bonilla, L.J. Yebra, and S. Dormido. Chattering in dynamic mathematical two-phase flow models. *Applied Mathematical Modeling*, 36:2067–2081, 2012.
- G.E.P. Box and N.R. Draper. *Empirical Model-Building and Response Surfaces*. Wiley, 1987.
- F. Casella, M. Otter, K. Proelss, C. Richter, and H. Tummescheit. The Modelica Fluid and Media library for modeling of incompressible and compressible thermo-fluid pipe networks. In *Proceedings of the 5th Modelica Conference*, 2006.
- L. Cecchinato and F. Mancini. An intrinsically mass conservative switched evaporator model adopting the moving-boundary method. *International Journal of Refrigeration*, 35:349–364, 2012.
- J.M. Corberan, I. Martinez-Galvan, S. Martinez-Ballester, J. Gonzalez-Macia, and R. Royo-Pastor. Influence of the source and sink temperatures on the optimal refrigerant charge of a water-to-water heat pump. *International Journal of Refrigeration*, 34:881–892, 2011.
- H. Elmqvist, H. Tummescheit, and M. Otter. Object-oriented modeling of thermo-fluid systems. In *Proceedings of the 3rd Modelica Conference*, 2003.
- R. Franke, F. Casella, M. Sielemann, K. Proelss, M. Otter, and M. Wetter. Standardization of thermo-fluid modeling in Modelica.Fluid. In *Proceedings of the 7th Modelica Conference*, 2009.
- C. Laughman. A comparison of transient heat pump cycle simulations with homogeneous and heterogeneous flow models. In *International Refrigeration and Air Conditioning Conference at Purdue University*, 2014.
- P. Li, Y. Li, J.E. Seem, H. Qiao, X. Li, and J. Winkler. Recent advances in dynamic modeling of HVAC equipment. Part 2: Modelica-based modeling. *HVAC&R Research*, 20(1):150–161, 2014a.
- P. Li, H. Qiao, Y. Li, J.E. Seem, J. Winkler, and X. Li. Recent advances in dynamic modeling of HVAC equipment. Part 1: Equipment modeling. *HVAC&R Research*, 20(1):136–149, 2014b.
- Modelon AB. *Air Conditioning Library User Guide*, 2015. v1.9.0.

EPSILON Modelica library for thermal applications

Laurent Lachassagne¹ Arnaud Colleoni¹ Hervé Feral¹ Nicolas Dolin¹

¹Epsilon Ingénierie, France, {llachassagne, acolleoni, hferal, ndolin}@epsilon-alcen.com

Abstract

This paper presents the Modelica library built by the French company “Epsilon Ingénierie” in order to provide system models of several technologies for thermal applications. The Epsilon library has its own structure for media definition, allowing simulation of two-phase phenomena in the library models. This library also includes several heat transfer technologies models such as heat exchangers, thermo-electric generators, heat pipes, loop heat pipes, etc... This paper presents two examples of system modeling with the Epsilon library using OpenModelica: a capillary pumped loop and a Fresnel solar plant.

Keywords: Modelica library, heat transfer, systems, medium, two-phase, solar

1 Introduction

The current trend in industry is to electrify and downsize gradually the mechanical and hydraulic technologies. This evolution can only succeed by managing the excessive heat that needs to be evacuated. As the thermal systems have to be more efficient using smaller exchange areas to meet industry needs, designing the thermal architecture of any system becomes more and more difficult with average heat transfer technologies. New promising technologies, such as two-phase heat transfer devices which are innovative heat transport systems with high efficiency, need to be studied and implemented in engineering phases. A variety of physical conditions, such as the working fluid, have also to be studied. And eventually, smart meters need to be integrated to be able to compare all the available solutions.

Under these circumstances, manufacturers and system designers need easy-to-use modeling tools in order to include all these new technologies inside global models, modify the physical conditions, and to assess their potentials. Epsilon has chosen Modelica to develop its own heat transfer technologies models, enriched by its background in spatial, aeronautic, and process thermal challenges over the years. The main objective is to use these models for any applications where global thermal management is needed, such as planes, district heating or power plants for example.

2 EPSILON library

The Epsilon library (Figure 1) includes two main subsections: “Media” where materials are defined and “Systems” where models are located. The other packages are “toolboxes” for users. They gather several useful components for system models and mostly inspired by Epsilon from the ModelicaStandard library, especially the FluidHeatFlow library. Thus, Epsilon library connectors design is the same as in the FluidHeatFlow library. The objective is to make all components of the Epsilon library compatible with components of the ModelicaStandard library and this connector design seems to be adapted to thermo-hydraulic modeling.

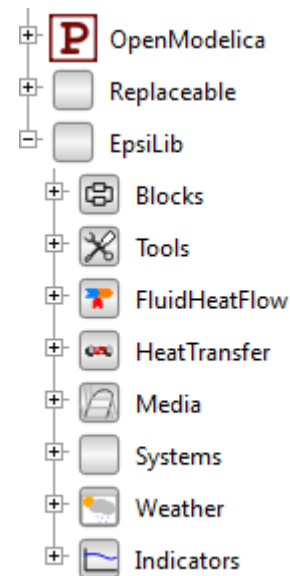


Figure 1. Structure of Epsilon library.

2.1 “Media”

In order to develop modular models, able to model phase change, where materials will be easily replaceable, Epsilon has decided to develop its own structure (Figure 2). This Media library is filled with material data and properties functions gathered by Epsilon over time. All materials (solids, fluids or gases) are called the same way: *EpsiLib.Media.Name.Phase.Temperature dependence (only for fluids).Type of laws.Properties functions*. This makes easier the

change of material in the studied models by modifying *Name*.

Regarding to fluids, models need only mass enthalpy and pressure as inlet variables. Temperature and vapor quality are computed inside fluid model. This way two-phase flows can be managed in systems models, such as the Capillary Pumped Loop presented in section 3.

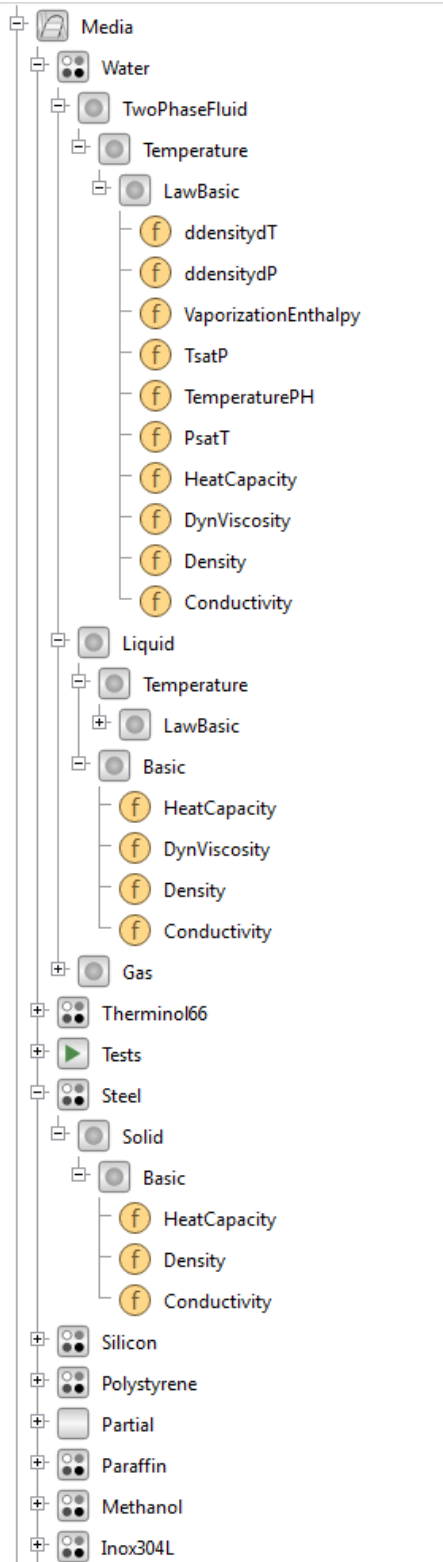


Figure 2. Structure of Epsilon “Media” library

The user can also choose to work with only one fluid state, by choosing “Liquid” or “Gas” instead of “TwoPhase” in the medium call *Phase*, to avoid non-linearity or to use lightest models. For the moment Epsilon has not encountered applications with multicomponent mixture media.

2.2 “Systems”

The Epsilon “Systems” library (Figure 3) is divided into several packages depending on the technical applications of thermal management Epsilon has already encountered and simulated. “ThermalStorage” contains Phase Change Materials models. “Solar” includes models and sub-models for designing solar power plant as the Fresnel solar plant described in section 4. “HeatExchanger” gathers different models of heat exchangers, evaporators and condensers. “Electrical” contains Thermo-electric Generators models (such as Peltier cells). “Diphasic” gathers systems where two-phase fluids rule, such as heat pipes, loop heat pipes, heat pumps, etc... Eventually, “Building” package includes models for buildings energy management. These packages are not exhaustive and the Epsilon library is defined to be filled over time with applications coming from Epsilon expertise or client needs.

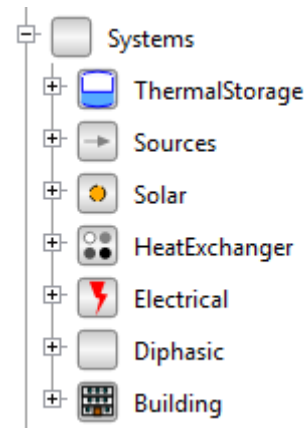


Figure 3. Structure of Epsilon “Systems” library.

2.3 Implementing indicators

When designing a thermal system, modeling has to supply answers to help choose between several configurations. To do so, several smart meters have to be included. Based on the “extends” possibilities of Modelica, the creation and integration of smart meters in system modeling is simplified.

Hence, a Modelica block has been created, containing the necessary equations to calculate costs at each time step with the following parameters:

- P0, initial cost of purchase
- P1Active, a Boolean when using P1, representing the cost of running energy
- P2, maintenance cost

- TimePeriod, for the maintenance time frequency
- P3, renewal facilities cost

To integrate a cost in a model, one only need to call the cost block “extends CostBlock(P0=xx, P1Active=true, P2=xx, P3=xx)” and to add an equation relative to P1. Moreover, a small icon with the Euro symbol will be added on the diagram view, allowing for a connection of the cost of each component to be treated in an economic model (as illustrated in figure 17).

Eventually, the generic structure of this meter makes it easy to create other indicators such as, CO₂ generated, size, mass, footprint ... Hence, models developed in the Modelica Library can support simulations over multiple time scales: seconds to minutes for local dynamics/controls studies and days or years for overall economic value propositions studies. An example using these cost indicators is given in section 4 of this paper concerning modeling of a solar Fresnel power plant.

3 Example: Capillary Pumped Loop model

A first example of using the Epsilon library is the model of a complex innovative system: a capillary pumped loop. This system is mostly used to transfer heat from sources with high heat fluxes densities to a cold source. It was first developed around fifty years ago for space applications (Stenger, 1966) to transfer heat from dissipating cells to solar panels. For around ten years, these systems are also used for terrestrial applications (railway, planes) such as power electronics cooling (Vasiliev *et al.*, 2009).

3.1 Principle

Capillary pumped loop (CPL) or Loop Heat pipes (LHP) are two-phase devices based upon the heat pipe working principle (Maydanik, 2004). The phase change of a pure fluid is used to transfer heat from the hot spot (evaporator) to the cold source (condenser). A porous media is inserted inside the evaporator to act like a pump for the fluid. The fluid flows in the loop due to combination of capillary pumping of liquid inside the porous wick and evaporation at the top of the wick. Liquid and vapor flow in separated lines between evaporator and condenser. A reservoir is added to the system to ensure that the evaporator wick will always be fed with enough liquid. The position of the reservoir determines the kind of loop and can impact the system behavior (Figure 4).

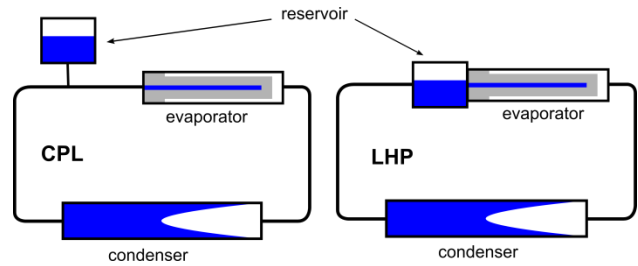


Figure 4. CPL vs. LHP.

The device used for this study is a particular kind of CPL: the capillary pumped loop for integrated power (CPLIP) developed by the Belgian company “Euro Heat Pipe”. Its particularity is the reservoir position, which is located above the evaporator (Figure 5). Then gravity has an influence on reservoir/evaporator coupling, making this design halfway between CPL and LHP. This system has been tested for a terrestrial application and results are available in literature (Lachassagne *et al.*, 2012). Epsilon society has developed and validated a Modelica system model of this device.

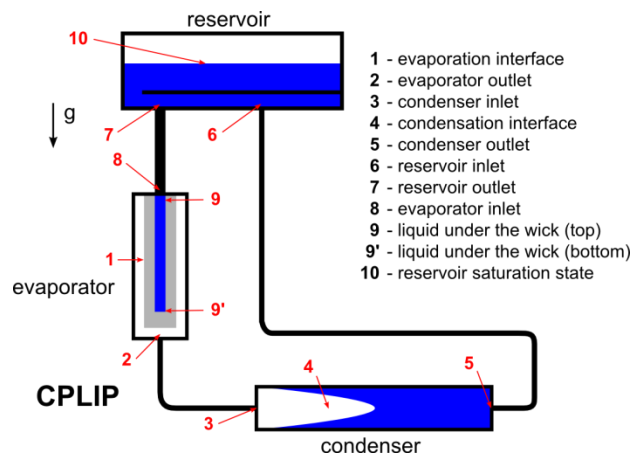


Figure 5. View of a Capillary Pumped Loop for Integrated Power (Lachassagne *et al.*, 2012).

3.2 Elementary cell

The fluid generic volume (Figure 6) is the key component of Epsilon two-phase and one-phase device models. This “generic pipe” model is fitted with two fluid connectors (inlet and outlet) and one heat connector (heat exchange through the volume frontier). Variables exchanged by the “flowPorts” are mass enthalpy, mass flow, enthalpy flow and pressure. The variables exchanged by the “heatPort” are heat flow and temperature.

This elementary model has been tested using fluids of Epsilon library “Media” with a set of boundary conditions (Figure 7). It works both for one-phase and two-phase fluid flows. Flow change of direction is also possible. Several heat transfer and pressure losses correlations are available in the Epsilon library package “Tools” and can be chosen by the user.

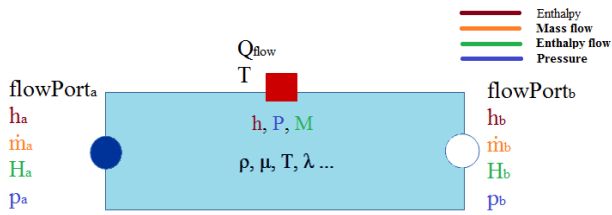


Figure 6. View of the elementary fluid volume variables.

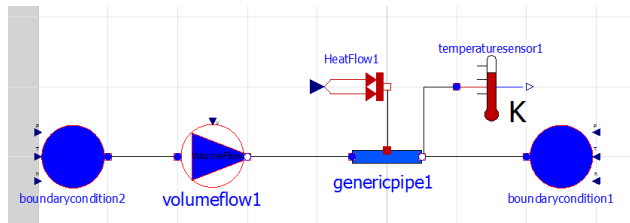


Figure 7. Test model of the elementary fluid volume “genericpipe” of the Epsilon library.

3.3 CPL system Model

By connecting several components of the Epsilon library, a complete CPLIP system model can be developed (Figure 8). According to the structure of Epsilon media library, this model allows to change working fluid and materials with only one clic. Loop geometry parameters and heat transfer correlations can also be changed by the users.

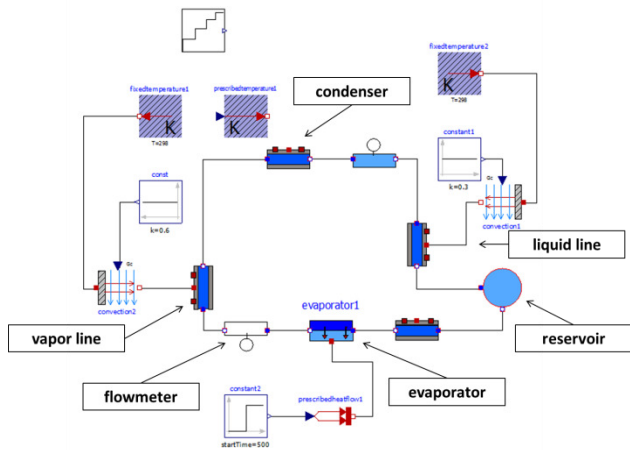


Figure 8. Test model of the CPLIP in the Epsilon library.

3.4 Validation

This model has been validated at steady-state by comparing with literature results (Lachassagne *et al*, 2012). These results were experimental data of a CPLIP test bench with ethanol as working fluid. Two kinds of models have been tested: one with a “complex” evaporator with many thermal and fluid couplings and the other “simplified” with only one coupling for the evaporation interface. On Figure 9 and Figure 10, dots stand for experiments results, dotted

lines stand for the simplified evaporator model and plain lines stand for the complex evaporator models. It appears that both model fit the experiments results well, with little more precision for the complex evaporator model. The error remains greater for the hydraulic variables than for the thermal ones, which is still acceptable considering the great instabilities due to condensation in these systems.

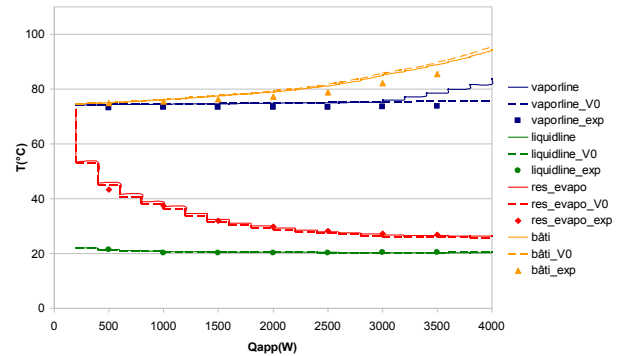


Figure 9. Temperatures vs. Heat power applied at the CPLIP evaporator.

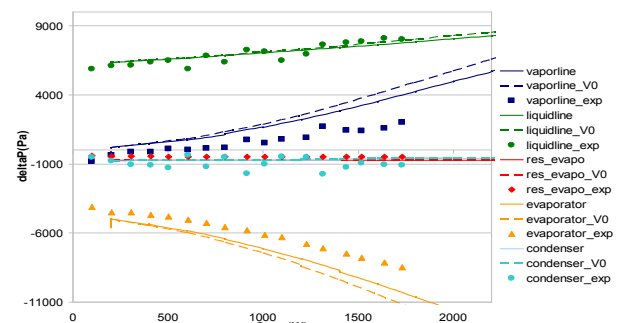


Figure 10. Pressure losses in the CPLIP components vs. Heat power applied at the CPLIP evaporator.

After comparing efficiency of these two CPL models, the last tests performed were numerical tests. Figure 11 shows the simulation duration function of number of elementary cell in the condenser. The objective of these tests was to assess the numerical reliability of the model not only for different complexity (two kinds of evaporator), but also for different model sizes (condenser discretization). Blue line stands for tests of the complex evaporator model and yellow line stands for the simplified evaporator model. Dots appearing on the abscissa axis represent a simulation crash for the corresponding number of condenser cells. The simplified model shows a quadratic behavior whereas complex evaporator model seems to have an exponential increasing of simulation duration with the condenser discretization. It also appears that complex model has many simulation crashes contrary to simplified one. Tests have also shown that the stability of the complex CPL model will be increased if the user

is particularly careful of the boundary and start conditions of the model.

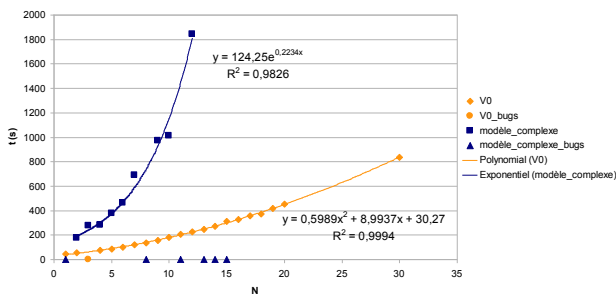


Figure 11. Simulation duration vs. CPLIP condenser cells number.

It remains hard to explain this numerical behavior because of complexity of these kinds of models couplings. It is well-known that phase-change modeling leads to large discontinuities in fluid properties and then numerical issues. This will be investigated by Epsilon in the future, based on literature analysis (Bonilla *et al*, 2012). The fluid flow connector design can also be discussed. Epsilon has chosen to use the Modelica FluidHeatFlow connector design, as the Fluid library was not compatible with OpenModelica when the library development started, but the use of stream connectors, (Franke *et al*, 2009) which seems more robust, will be investigated for the future versions of Epsilon library.

To conclude, the interest for Epsilon of using Modelica for this kind of modeling is its modularity. It is really easy to have many levels of complexity for the system models and the user can choose what fits better with his objectives: more precision but less stability or less precision but more stability and short simulation durations.

4 Example: Fresnel solar power plant

A second example of using the Epsilon library is the model of a solar power plant.

4.1 Principle

Fresnel solar power plants (Figure 12) are linear thermal solar concentrators formed by an assembly of flat mirrors named “compact linear reflectors”. Each reflector can spin according to sun location in order to reflect and concentrate sunbeams to one or more fixed receiver pipes. The fluid flowing through these pipes can then turn into vapor up to 500°C. This vapor is available for industrial process or electricity generation for instance.

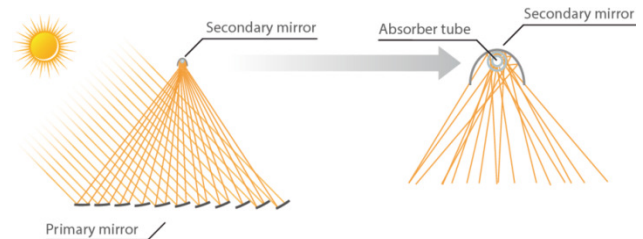


Figure 12. Linear Fresnel solar power plant principle.

Some studies about Fresnel solar power plants propose modeling of reflectors behavior (Pino *et al*, 2013). One study in particular has been performed using Modelica in order to simulate working points of the Fresnel solar power plant developed by Alsolen society (Rodat *et al*, 2014). This model still suffers some limitations, such as no storage modeling or also calling an external simulation code for mirrors modeling.

4.2 Components

4.2.1 Receiver

The power plant receiver has been developed using the elementary fluid volume described in Figure 6. The power plant receiver is subjected to three major thermal phenomena: solar heat flux absorption, radiations to environment and convection losses. These phenomena appear in the receiver detailed model developed by Epsilon in its library (Figure 13).

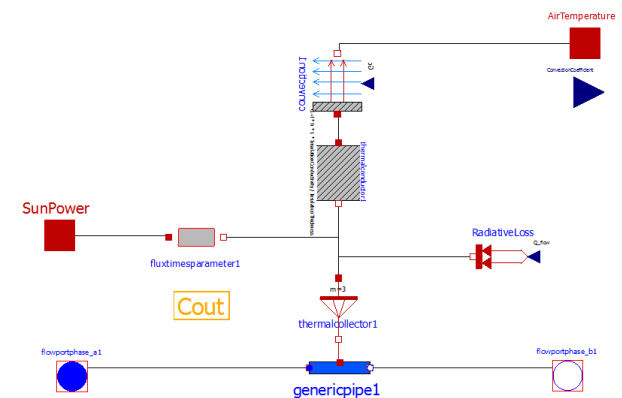


Figure 13. Receiver Modelica model structure.

The receiver component (Figure 14) is fitted with six ports:

- two fluid ports,
- one “heatport” for air temperature and one real port for convection coefficient, both depending on weather conditions,
- one “heatport” for received solar heat flux,
- one real port “Cost” for receiver cost transmission to global models.

Like all Epsilon library components, this receiver has been validated by simple test models.



Figure 14.Receiver component design in Epsilon library.

4.2.2 Mirrors

A component describing optical behavior of a Fresnel mirror has been developed thanks to literature studies (Pino *et al*, 2013). This component can take into account the receiver shade on the mirrors field and the shade of mirrors to each other. This component appears in Figure 15 and is made of:

1. Weather data.
2. Solar angle calculation
3. Fresnel mirror.
4. Shade calculation.
5. Reflected solar heat flux.
6. Mirrors field cost calculation.

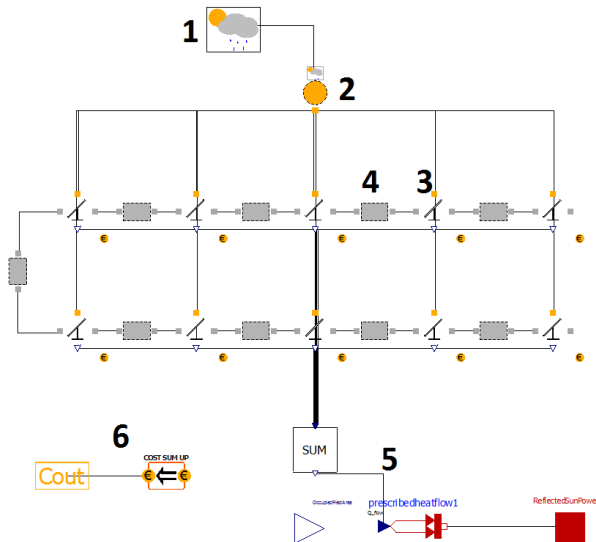


Figure 15.Mirror field assembly modeling.

4.3 Solar power plant model

The Fresnel solar power plant of this study is similar to the one has been developed by Alsolen company (Figure 16). Its features are:

- A 1000 m² solar field,
- Therminol66 as working fluid,
- rock-bed thermal storage,
- a secondary heat loop by organic Rankine cycle.

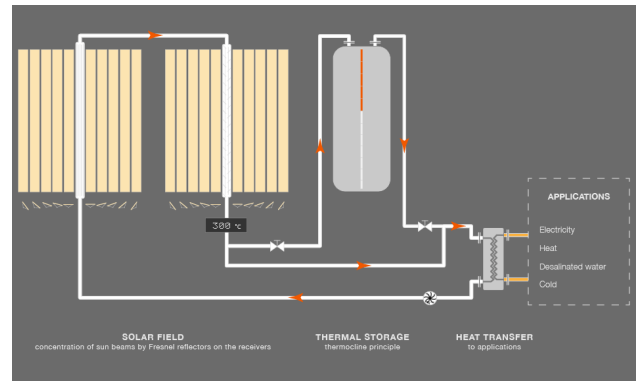


Figure 16.Linear Fresnel solar power plant of Alsolen company.

One main hypothesis has been made in the Epsilon modeling of this plant (Figure 17): heat absorbed by the receiver working fluid is totally exchanged to the secondary loop which is currently not taken into account in the global model.

The components of the solar power plant are then:

1. Weather data, using classical weather files from building codes.
2. Solar receiver.
3. Fresnel mirrors field.
4. Thermocline thermal storage.
5. Mass flow rate regulation.
6. Total cost calculation.

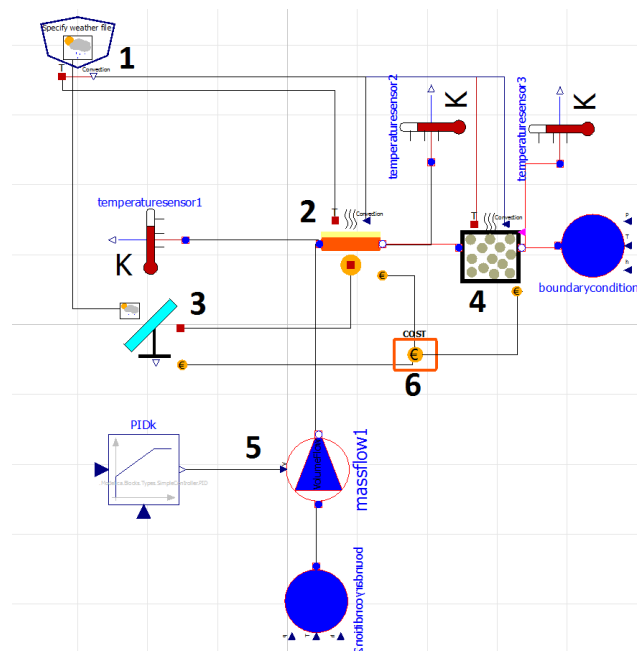


Figure 17.Epsilon Modelica model of the Alsolen Fresnel solar power plant.

4.4 Results

The solar power plant has first been tested for three different locations in France (Table 1): Rouen (north of France), Toulouse (south of France) and Perpignan (extreme south of France).

Table 1. Solar power plant efficiency at three locations.

location	Rouen	Toulouse	Perpignan
Solar resource (kWh/m ² /year)	905	1261	1441
Reflected heat flux (kWh/m ² /year)	733	1033	1188
Optical efficiency	0.81	0.82	0.82
Absorbed heat flux (kWh/m ² /year)	436	661	772
Thermal efficiency	0.59	0.64	0.65
Cost / exergy (€/kWh)	1.26	0.88	0.75

Those first results allow checking the order of magnitudes obtained. Logically, Rouen is less favorable than Perpignan for the implantation of such central, decreasing the absorbed heat flux and the thermal efficiency (as the external temperature is lower). Eventually, the order of magnitude of the cost/exergy is realistic, with 75cts per kWh, based on an hypothesis on 20 years funding (Nixon *et al.* 2012).

The Epsilon Modelica model of the solar power plant has also allowed to test the impact of spacing arrangement of mirrors on the plant efficiency (Table 2) and the cost/exergy.

Table 2. Solar power plant efficiency function of mirror spacing arrangement.

mirrors spacing / mirrors width	0	0.5	1	2	3
Solar resource (kWh/m ² /year)	1261	1261	1261	1261	1261
Reflected heat flux (kWh/m ² /year)	901	1033	1056	1050	1030
Optical efficiency	0.71	0.82	0.84	0.83	0.82
Absorbed heat flux (kWh/m ² /year)	560	661	678	664	642
Thermal efficiency	0.62	0.64	0.64	0.63	0.62
Cost / exergy (€/kWh)	0.85	0.88	0.99	1.32	1.59

The spacing arrangement of mirrors is optimal when equal to the mirror width, when looking at the reflected

heat flux. A smaller spacing arrangement creates shadowing between mirrors whereas a bigger spacing implies optical losses due to the angular position of mirrors to aim at the receiver.

The cost/exergy analysis gives a different result, where the optimal spacing between mirrors should be null, to minimize the ground size occupied (and so its cost). However, this solution is unrealistic since for maintenance and cleaning reasons, it is necessary to be able to circulate between the mirrors.

5 Conclusion

Epsilon has developed its own library of thermal solutions modeling with Modelica. This library is based upon a proper structure with media calculation allowing to simulate phase change phenomena well as integrating indicators (such as cost calculation) in global system models. Some models have already been developed and validated, such as capillary pumped loop model. This library provides several kinds of components which can be used for global system modeling, such as power plant modeling. Epsilon will continue to add new components for thermal control with the maximum return of experience available.

References

- Yuri F. Maydanik, Loop Heat Pipes, *Applied Thermal Engineering*, No 25, pp. 635-657, 2004.
- Laurent Lachassagne, Vincent Ayel, Cyril Romestant and Yves Bertin. Experimental study of capillary pumped loop for integrated power in gravity field. *Applied Thermal Engineering*, No 35, pp. 166-176, 2012.
doi:10.1016/j.applthermaleng.2011.10.019
- F.J. Stenger, Experimental Feasibility Study of Water-filled Capillary-pumped Heat-transfer Loop, NASA TM X-1310, Lewis Research Center, Cleveland, OH, 1966.
- Leonid Vasiliev, David Lossouarn, Cyril Romestant, Alain Alexandre, Yves Bertin, Yauheni Piatsiushyk and Vladimir Romanenkov. Loop heat pipe for cooling of high-power electronic components. *International Journal of Heat and Mass Transfer*. No 52, pp. 301-308, 2009.
- J. Bonilla *et al.* Chattering in dynamic mathematical two-phase flow models, *Applied Mathematical Modelling*. No 36(5), pp. 2067-2081, 2012.
doi:10.1016/j.apm.2011.08.013
- R. Franke *et al.* Stream Connectors - an Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena, *Proc. 7th International Modelica Conference*, Como, Italy, Sep. 20-22, 2009, pp. 108-121.
- F. J. Pino, R. Caro, F. Posa and J. Guerra. Experimental validation of an optical and thermal model of a linear Fresnel collector system. *Applied Thermal Engineering*, No 50, pp. 1463-1471, 2013.

- S. Rodat, J. V. D. Souza, S. Thebault, V. Vuillerme and N. Dupassieux. Dynamic simulation of Fresnel solar power plant. *Energy Procedia*, No 49, pp. 1501-1510, 2014.
- J.D. Nixon and A. Davies, Cost-exergyoptimisation of linear Fresnel reflector. *Solar Energy* No 86, pp. 147-156, 2012.

Multi-objective optimization of dynamic systems combining genetic algorithms and Modelica: Application to adsorption air-conditioning systems

Uwe Bau¹ Daniel Neitzke¹ Franz Lanzerath¹ André Bardow¹

¹Institute of Technical Thermodynamics, RWTH Aachen University, Germany,
andre.bardow@ltt.rwth-aachen.de

Abstract

The Modelica language enables the fast and convenient development of physical simulation models. These models are often used for simulation studies. The re-use of simulation models for optimizations requires model-adaptions, additional tools or libraries. In this paper, we present a framework to connect Modelica models developed in Dymola to MATLAB's optimization toolbox. As optimization algorithm, we use a multi-objective genetic algorithm. The optimization procedure is tested for an adsorption air-conditioning design. Compared to a full factorial design, the optimization procedure produces better solutions using less evaluations.

Keywords: multi-objective, optimization, Pareto-resolution, dynamic systems, full factorial design, genetic algorithms, MATLAB, gamultiobj, NSGA-II, adsorption air-conditioning systems

1 Introduction

Modelica is an acausal, object-oriented and equation-based programming language with a high number of physical model libraries available. These language features allow for fast and convenient development of physical models. These models are used to enhance system understanding and performance by simulation and parameter variations. To go beyond parameter variations, optimization functionality is desired, e.g. for product design, parameter estimation or optimal control. To reduce development costs and effort, it would be desirable to employ simulation models directly for optimization.

The currently available options to transform a simulation model into an optimization model differ widely, starting from commercial Modelica libraries to freely available external tools. Several optimization tools are available specifically for Modelica:

For Dymola, the commercial library Design/Optimization exists, which includes different optimization algorithms, such as Sequential Quadratic Programming or Genetic Algorithm (Pfeiffer, 2012).

The open-source project JModelica provides Optimica, an extension of the Modelica language. Optimica allows for high-level formulation of optimization problems using modelica models (Lind et al., 2012; Dietl et al., 2014).

OMOptim is an optimization framework based on the open source OpenModelica platform. OMOptim includes meta-heuristic optimization algorithms such as genetic algorithms and is under further development (Thieriot et al., 2011).

Alternatively, Modelica models can be connected to external optimization tools:

GenOpt is such an open source optimization framework, which can be used for any simulation tool allowing input file modification and output reading. GenOpt includes several optimization algorithms, such as Generalized Pattern Search or Particle Swarm Optimization (Wetter, 2000).

Modelon provides a commercial Functional Mockup Interface (FMI) for MATLAB's widely used optimization toolbox (Henningsson et al., 2014). This FMI allows Modelica models to be optimized with MATLAB.

FMI has also been used to connect Modelica models to connect via FMI to the optimization code MUSCOD-II for gradient-based dynamic optimization (Gräber et al., 2011; Leineweber et al., 2003).

Which optimization approach suits best, depends on the resources and experience of the developer. In this work, we present a convenient approach connecting MATLAB's optimization toolbox for multi-objective optimization tasks to existing Modelica models. Here we used Modelica models developed within Dymola using our LTT Adsorption Energy Systems Library (Bau et al., 2014). We coded the MATLAB-Dymola interface using Modelica script files (.mos) embedded within the MATLAB-code.

In this paper, we present the developed interface for a design case study of an adsorption air conditioning system for battery-driven busses. As optimization algorithm, we use a multi-objective genetic algorithm, which is robust and has few requirements regarding model char-

acteristics. The problem has 8 design parameters, which are optimized regarding 2 objective functions.

In Section 2, we describe the optimization algorithm used and the MATLAB/Dymola interface. Section 3 contains the non-linear dynamic process model used for the design task. In Section 4, we discuss the obtained optimization results and compare the genetic optimization algorithm with a full factorial design. Finally, we summarize our findings in Section 5.

2 Optimization procedure

This paper studies the optimization of conflicting key performance indicators. Therefore, we define the resulting multi-objective optimization problem in Section 2.1. In Section 2.2, we present the applied optimization algorithm followed by the framework to interact with Modelica in Section 2.3. The handling of infeasible solutions and unsatisfied path-constraints is discussed in Section 2.4.

2.1 Multi-objective optimization problem

A designer often seeks to optimize several conflicting key performance indicators of a product by varying independent design parameters. Each parameter has usually a certain feasible range. Furthermore, feasible solutions often need to satisfy additional constraints (e.g. physical or operational limits).

The n_O relevant key performance indicators are here called objectives O with

$$O_i = f_i(x, z, p), \quad i = 1, \dots, n_O \quad (1)$$

as a function of differential states $x(\tau)$, algebraic states $z(\tau)$ and independent design parameters p . τ represents time with

$$\tau \in [0, T] \quad (2)$$

The n_p design parameters or decision variables p are restricted to a feasible decision or solution space $S \in \mathbb{R}^{n_p}$ within the constraints or bounds $[LB, UB]$. Furthermore, the differential and algebraic states x and z are required to satisfy so called path-constraints $LB^* \leq x, z \leq UB^*$ at any given moment in time τ .

The problem can be mathematically defined as:

$$\begin{aligned} & \min_{x, z, p} O & (3) \\ \text{s.t.} & \dot{x} = f(x, z, p) & (4) \\ & 0 = g(x, z, p) \\ & LB \leq p \leq UB \\ & LB^* \leq x, z \leq UB^* \quad \forall \tau \\ & x(\tau = 0) = x_0 \\ & p \in S, S \in \mathbb{R}^{n_p} \end{aligned}$$

The designer is concerned about the global optimum. When two solutions are compared regarding their objective values, the solution $O_{\text{dom}} = f(p_{\text{dom}})$ is said to dominate solution $O_{\text{inf}} = f(p_{\text{inf}})$, if none of the objectives $O_{i, \text{dom}}$ is worse than $O_{i, \text{inf}}$ and at least one objective $O_{j, \text{dom}}$ is strictly better than $O_{j, \text{inf}}$.

The set of solutions that are not dominated by any other solution in the entire feasible solution space S are called Pareto-optimal.

2.2 Optimization algorithm

The dynamic system studied in this paper consists of complex, non-linear and spatial discrete differential equations. The differentiation of the objectives O with respect to the design parameters p can be complicated and time consuming, especially for dynamic and complex systems. Deterministic gradient-based optimization algorithms also require good initial guesses. The relatively high number of 8 design parameters and possible combinations complicates this approach.

In this paper, we therefore use a genetic algorithm, belonging to the group of derivative-free or heuristic algorithms. Other possible heuristic algorithms are reviewed in Konak et al. (2006) or Jones et al. (2002).

A genetic algorithm tries to imitate the evolutionary process of nature. It interprets solutions as individuals and design parameters as their genes, which are passed, mixed and mutated from parents to a new generation of children. Ultimately, only the fittest individuals with most optimal objective values “survive” and are chosen as new parents.

In case of incomplete or imperfect information on the behavior of the objective functions or limited computation capacity, stochastic meta-heuristics like genetic algorithms offer a satisfying approach to analyze the complex correlation of the design parameter’s influence on the objective values. Especially in case of a highly non-convex shaped solution space S , gradient-based algorithms are susceptible to converge to local optima (Blum and Roli, 2003). In contrast, heuristic approaches with random initialization and spontaneous mutation of genes allows to move the population away from local optima (Tomoiagă et al., 2013).

According to Audet and Vicente (2008), a genetic algorithm generally has a tendency to rapidly converge to generally “good” solutions in early iterations and predominantly “smooth” the distribution of Pareto-solutions in later iterations. Therefore, a genetic algorithm is best suited to successfully address an optimization problem for cases in which three major aspects apply: First, the differentiation of the objective functions within the bounds of the domain is expensive and similar approaches to approximate the derivatives (e.g. by finite-difference) are prohibited. Second, the objective functions are not excessively non-smooth (e.g. event-based binary factorization). And third, finding a “good” lo-

cal optima fast is significantly more important than definitely converging to the global optimum.

Several different software packages or libraries contain heuristic optimization methods, such as the global optimization toolbox for MATLAB. We use MATLAB's *gamultobj*-function, which is based on a variant of the Non-dominated Sorting Genetic Algorithm-II (NSGA-II). The NSGA-II not only favors elite, but also diverse solutions and therefore covers a wide range of the solution space S (Deb, 2001). The *gamultobj*-function itself offers a set of variable algorithm options to customize key randomization properties as well as termination conditions regarding computation time.

Genetic algorithms generate a fixed number of n new parameter sets to be evaluated at each generation. These parameter sets can be split for parallel processing on multi-core computers, which can reduce computation significantly. However, since both algorithms applied in this paper (genetic and full-factorial) can be processed parallel, this topic is not discussed any further.

2.3 Optimization framework

Our implementation employs current versions of both MATLAB (R2014a) and Modelica (2014 v3.2). The model of the presented case study in this paper is based on Modelica models from our LTT Adsorption Energy Systems Library (Bau et al., 2014).

A framework to connect *gamultiobj* to existing Modelica models in Dymola is implemented, so that no further commercial optimization program is required. *gamultiobj* treats the Modelica model as a black-box. *gamultiobj* sets parameters as input values for the Modelica model, which produce objectives as output values.

A flow chart of the framework's procedure is displayed in Figure 1 and can be described as followed. The procedure first calls

$$p = \text{gamultiobj}(f_{\text{fitness}}, n_{\text{vars}}, A, b, \dots, A_{\text{eq}}, b_{\text{eq}}, LB, UB, \text{options}) \quad (5)$$

which randomly selects a set of parameters p as initial population (1 → 2). The (in-)equality constraints and parameter bounds satisfy the following equations:

$$A \cdot p \leq b \quad (6)$$

$$A_{\text{eq}} \cdot p = b_{\text{eq}} \quad (7)$$

$$LB \leq p \leq UB \quad (8)$$

However, we only use Equation 8 and set $A = b = A_{\text{eq}} = b_{\text{eq}} = []$, the empty matrix. In each iteration, *gamultiobj* calls a fitness-function

$$O = f_{\text{fitness}}(p) \quad (9)$$

The fitness function usually contains the models equations describing the correlation between the objective O

and the parameters p . Here, we call a Modelica script file (.mos), which executes the Dymola command *simulateExtendedModel()* while passing the parameter set p (2 → 3). The objective values are stored in a MATLAB data file (.mat) and evaluated in f_{fitness} . Finally, the resulting objective values of each solution are returned as output variables of f_{fitness} to *gamultiobj* (3 → 4). The genetic algorithm then processes the fitness and diversity of the solution and selects the best (elite) and most diverse solutions as the parent individuals of the next generation (4). Due to random recombination, mutation and migration of individuals (5), a new set of parameters p is generated and the iteration starts over until a termination criteria applies (6 → 7, 2). For example, one criteria can be the exceeding of the maximum generation number.

In order to apply this procedure for any given optimization problem for an existing Modelica model, the user only has to define the design parameters and their corresponding bounds, as well as the objective functions. Neither the model, nor the algorithm needs to be altered to suit a tailor-made optimization. Even the modification of the algorithm options in *gamultiobj* is optional.

One such option is the population size, which is constant for each generation. This can lead to the loss of Pareto-solutions over generations. Therefore, we store every Pareto-set of each generation in a separate file to keep all elite solution.

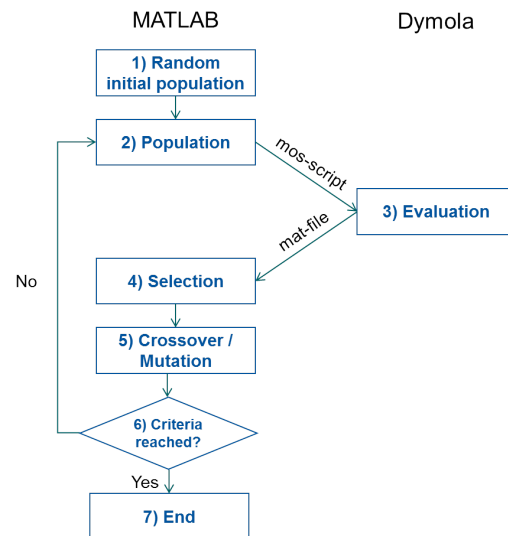


Figure 1. Procedure of optimization with genetic algorithm using MATLAB's global optimization toolbox and Dymola for evaluation (system simulation and objective calculation).

2.4 Handling infeasible solutions and unsatisfied path-constraints

The parameter constraints $LB \leq p \leq UB$ are satisfied by *gamultiobj* by choosing only feasible parameter values p for a simulation. However, the satisfaction of path-constraints and the feasibility of the simulation (e.g. due

to unsolvable stiff differential equations) depend on the choice of parameters and can only be checked after each simulation. Therefore, we implement an evaluation stage in MATLAB, which adds a penalty function PF to the objective function O :

$$O^* = O + PF \quad (10)$$

The penalty function PF is a piecewise function

$$PF = \begin{cases} 0, & LB^* \leq x, z \leq UB^* \\ \infty, & \text{else} \end{cases} \quad (11)$$

Infeasible solutions and unsatisfied path-constraints lead to positive infinite objective values and are therefore unfavored by the minimization problem. Alternative means of implementing penalty functions are possible (e.g. continuous penalty function or penalty function as separate objective value), but are not discussed in this paper.

3 Case Study: Adsorption-storage air-conditioning system

An adsorption air-conditioning system for electrical busses serves as representative case study of a dynamic and complex system in this paper. For electric busses, the air-conditioning takes up to 50% of the battery capacity when using a battery-driven compression chiller (Bottiglione et al., 2014). The adsorption air-conditioning (AC) system aims at providing cooling and heating. Since thermal energy is stored as latent and adsorption enthalpy, operating the AC system requires only a small of the battery capacity for ventilation. In the following Sections 3.1 - 3.4, the cooling mode is presented. For additional information see also (Bau et al., 2015).

3.1 Concept

The presented adsorption air-conditioning concept is based on the Pennington cycle (Pennington, 1955). During a bus ride, the adsorption AC system provides cooling and dehumidification (see Figure 2):

- Bus air (green arrows) flows through the desiccant module, which contains the sorbent material. In the desiccant module, moisture is adsorbed and hot and dry air leaves the module (4) → (5).
- The hot and dry air is cooled by a heat exchanger (5) → (6). As coolant stream, ambient air (blue arrows) is used, which is cooled by evaporative cooling (1) → (2) before it enters the heat exchanger and takes heat from the hot and dry bus air (2) → (3).
- The bus air (6) is finally cooled down to the desired temperature by evaporative cooling (6) → (7).

Within this study, the AC system is optimized assuming a dry adsorption module at beginning of bus ride.

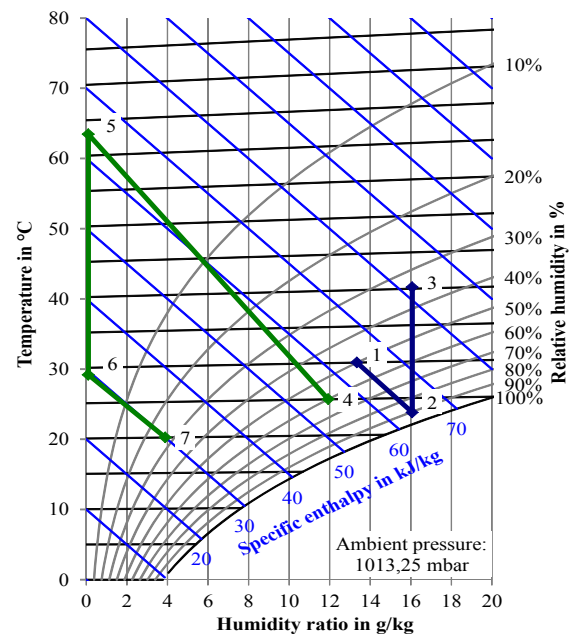
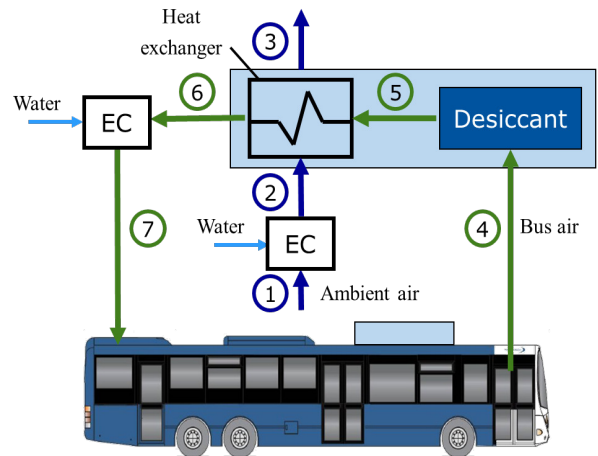


Figure 2. Cooling mode of adsorption storage: Process scheme (top) and Mollier diagram (left).

3.2 Model

The main parts of the model are the desiccant module and the heat exchanger (see Figure 3). Within these models, the air volume, the heat exchanger plates and the adsorbent are coupled by heat and mass transfer models. The Modelica model is built by using the LTT Adsorption Energy Systems library (Bau et al., 2014) and the TLK TIL library (Gräber et al., 2010). For further information regarding the heat and mass transfer correlations used, see Bau et al. (2014).

3.3 Key performance indicators (objectives)

The system performance can be quantified by two performance indicators: Specific cooling power (SCP) and coefficient of performance (COP). The specific cooling power (SCP) measures the power density by the ratio of

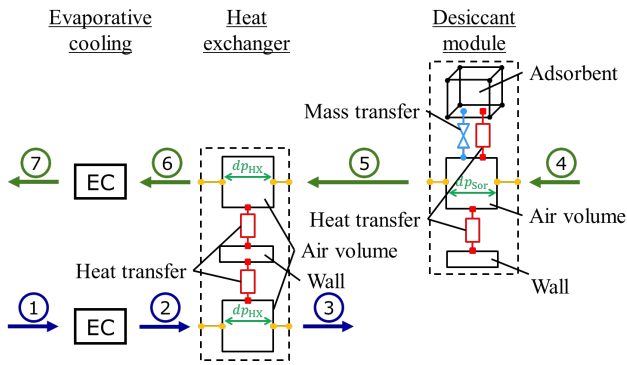


Figure 3. Structure of the dynamic model: Evaporative cooling (left), heat exchanger (middle) and desiccant module (right). The air flows correspond to Figure 2.

the average cooling power during a bus ride $\bar{Q}_{cooling}$ to the mass of the system m

$$SCP = \frac{\bar{Q}_{cooling}}{m} \quad (12)$$

The mobile coefficient of performance (COP_{mobile}) measures the efficiency by the ratio of the average cooling power $\bar{Q}_{cooling}$ to the average electric power consumption $\bar{P}_{battery}$ during the busride:

$$COP_{mobile} = \frac{\bar{Q}_{cooling}}{\bar{P}_{battery}} \quad (13)$$

$\bar{P}_{battery}$ is required only for ventilation in the adsorption-based AC system.

3.4 Design parameters

As design case, we choose a hot summer day in Germany with ambient temperature $T_{ambient} = 30^\circ\text{C}$ and humidity $\phi_{ambient} = 50\%$.

The system is optimized for $n_p = 8$ design parameters p with lower and upper bounds: 5 heat exchanger design parameters, 2 adsorber parameters and the ratio between bus and ambient air mass flow rates. All design parameters are listed in Table 1.

4 Results

In order to properly assess the benefit of a genetic optimization algorithm, we first introduce the results of a simple full factorial design as benchmark. In Section 4.2, we evaluate the development of the Pareto solutions produced by *gamultiobj()* for increasing generation numbers regarding convergence and diversity. Finally, we discuss the case study results.

Table 1. Design parameters p of adsorption air-conditioning system

Design parameter	LB	UB	Unit
$length_{HX}$	0.1	2	m
$height_{HX}$	0.1	2	m
$width_{HX}$	0.1	2	m
$duct\ height_{bus}$	0.001	0.006	m
$duct\ height_{ambient}$	0.001	0.006	m
$diameter_{sorber}$	0.0005	0.008	m
$height_{adsorber}$	0.4	0.08	m
$\frac{\dot{m}_{ambient}}{\dot{m}_{bus}}$	0.5	5	-

4.1 Full factorial design

A full factorial design evaluates all possible combinations of each n_p factors or parameters and its k assigned levels or discrete states. If we assign to each parameter the same number of discrete states, the number of required experiments or simulations n_{sim} amounts to

$$n_{sim} = k^{n_p} \quad (14)$$

In our case study, we chose to assign to each of our $n_p = 8$ parameters its upper and lower bounds a and b and the arithmetic average as possible discrete states:

$$p = \left[LB, \frac{LB+UB}{2}, UB \right] \quad (15)$$

Therefore, $3^8 = 6561$ simulations in total are required. Since a single simulation of our model takes about 8 s of computation time, a full factorial design amounts to 14.5 h CPU-time in total. The results of all feasible solutions with positive values for both objectives are displayed in Figure 4 with the final Pareto solutions highlighted in red.

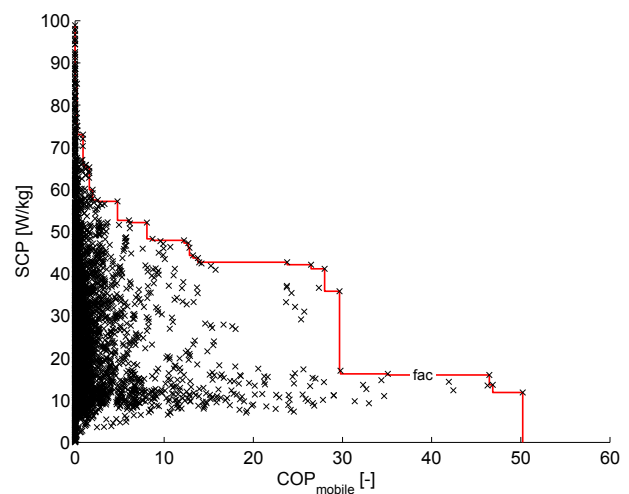


Figure 4. All data points of full factorial design (6561 simulations) and the obtained Pareto front (red line).

From the 6561 simulations, we receive 29 Pareto solutions within the maximum range of $COP_{mobile,max} = 50.2$ and $SCP_{max} = 98.9 \text{ W kg}^{-1}$. The Pareto solutions are not equidistantly distributed on either dimension. This results in occasional spacious gaps between the objective values of adjacent Pareto-solutions. If the designer aims to reach a certain benchmark or minimum limit for one objective, those gaps might force the designer to accept significant losses on the other objective.

4.2 Genetic algorithm

The Pareto solutions for selected generations of the genetic algorithm are displayed in Figure 5 in comparison to the Pareto solution of the full factorial design. Each generation consists of 164 simulations and therefore represents the equivalent computation time of approximately 2.5% of the complete full factorial design with 6561 simulations.

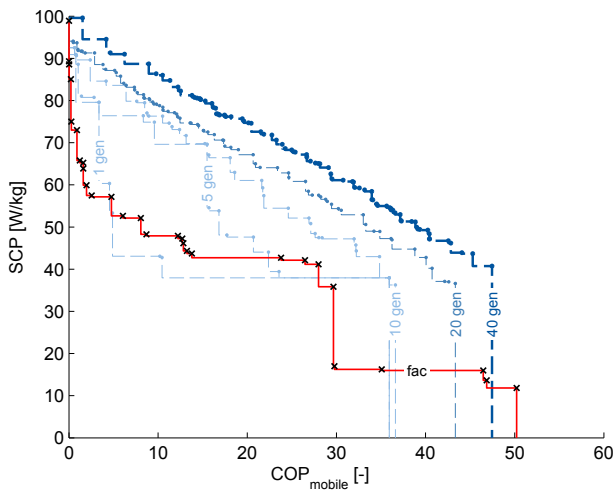


Figure 5. Development of Pareto solutions by the genetic algorithm (in shades of blue) compared to Pareto solutions by a full factorial design with the same number of simulations: $3^8 \approx 164 \cdot 40 = 6560$.

After 40 generations, the genetic algorithm produced 80 Pareto solutions within the maximum range of $COP_{mobile,max} = 48.0$ and $SCP_{max} = 99.6 \text{ W kg}^{-1}$. The range of Pareto solutions is comparable for the full factorial design and the genetic algorithm, however, the genetic algorithm produces more than twice as many solutions, which are more evenly distributed. Also, the maximum range of every objective increases for each generation, which supports the fact that the genetic algorithm favors diverse solutions and minimizes the risk of running into local optima. Thus, the designer has a higher resolution of solutions to choose from.

As predicted in Section 2.2, the genetic algorithm converges to relatively good solutions in early stages. The Pareto solutions of the genetic algorithm outperform all but 3 of those of the factorial design after 5 generations already. This corresponds to 12.5% of the respective

CPU-time of all factorial design simulation runs. Despite the fact that the initial guess is selected randomly, the algorithm converges quickly and can therefore be considered robust to initial conditions. To study its robustness with respect to the initial guess, the genetic algorithm was started several times using different initial populations leading to slightly different evolution paths and final Pareto-solutions respectively.

The genetic algorithm produces better Pareto solutions than the factorial design with generally higher objective values. The evolution of the Pareto-front over each generation implicates that the range of the solution space S can continue to expand. If the designer is interested in more “extreme” solutions, we suggest to continue the algorithm for additional generations.

In summary, the genetic algorithm can be favored over the factorial design regarding robustness, convergence, Pareto solution value quality and quantity. By implication, the necessary computation time can be significantly reduced to obtain comparable results.

4.3 Case Study Results

To interpret the optimization results, a benchmark is needed. As benchmark, we regard a conventional compression chiller for busses with a fixed specific cooling power $SCP = 100$ and a fixed coefficient of performance $COP_{mobile} = 1.9$ (Spheros GmbH, 2015). Compared to this conventional system, the optimization results of the adsorption air-conditioning system show a trade-off between SCP and COP_{mobile} . This trade-off enables the designer to choose solutions with high COP_{mobile} values and still reasonable SCP values. For example, the design point $COP_{mobile} \approx 20$ and $SCP \approx 80 \text{ W kg}^{-1}$ allows the designer to reduce the needed battery capacity by a factor of 10 while only losing 20% of specific cooling power. Which point is taken as optimal depends on the specific application; in particular, weight limitations preference for high SCP , whether battery costs preference for high COP .

The “smoothing” of the genetic algorithm in later iterations (see Section 2.2) has two consequences: First, adjacent Pareto solutions are more similar. Second, the number of produced Pareto solutions within the feasible solution range S is higher than with a full factorial design. These facts strongly favor the genetic algorithm over the full factorial design when analyzing parameter-objective correlations $O_j = f(p_i)$. One such correlation is displayed in Figure 6. In Figure 6 we analyze the correlation of the heat exchanger mass flow ratio as the design parameter $p_i = \dot{m}_{ambient}/\dot{m}_{bus}$ and the mobile coefficient of performance as the objective $O_j = COP_{mobile}$.

The Pareto solutions of the genetic algorithm (black curve) have a much higher resolution than those of the full factorial design (red), which enables the designer to identify correlations more easily and adjust the system’s performance more precisely.

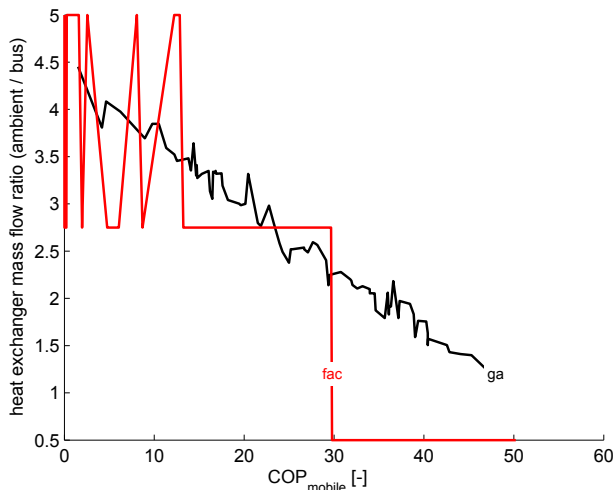


Figure 6. Change of the proposed heat exchanger mass flow ratio $\dot{m}_{\text{ambient}}/\dot{m}_{\text{bus}}$ for Pareto solutions of the genetic algorithm with ascending $\text{COP}_{\text{mobile}}$ values.

The correlation $\text{COP}_{\text{mobile}} = f(\dot{m}_{\text{ambient}}/\dot{m}_{\text{bus}})$ can be physically explained as followed: A higher ambient mass flow rate (higher $\dot{m}_{\text{ambient}}/\dot{m}_{\text{bus}}$) increases the amount of transferred heat, resulting in higher average cooling power \bar{Q}_{cooling} (higher SCP). As a trade-off, the effort for ventilation increases as well and raises the average electric power consumption \bar{P}_{battery} . The latter aspect dominates. In consequence, the mobile coefficient of performance $\text{COP}_{\text{mobile}} = \bar{Q}_{\text{cooling}}/\bar{P}_{\text{battery}}$ decreases for an increasing heat exchanger mass flow ratio $\dot{m}_{\text{ambient}}/\dot{m}_{\text{bus}}$. Other design parameters p_i can be analyzed in the same way.

5 Summary

In this paper, a procedure to connect Modelica models developed in Dymola to MATLAB's optimization toolbox is presented. The non-linear, dynamic process model is evaluated as a black box and the simulation results are loaded in MATLAB using Modelica script files. The framework is illustrated using a genetic algorithm to optimize the design of an adsorption air-conditioning system. The obtained optimization results are compared to a full factorial design: The optimization procedure outperforms the full factorial design regarding simulation time, solution diversity and objective values.

6 Acknowledgment

This work is funded by the Excellence Initiative of the German federal and state governments.

References

- Charles Audet and Luís Nunes Vicente. Derivative-Free Optimization: Theory and Practice, 2008.
- Uwe Bau, Franz Lanzerath, Manuel Gräber, Heike Schreiber, Niklas Thielen, and André Bardow. Adsorption energy systems library - Modeling adsorption based chillers, heat pumps, thermal storages and desiccant systems. In *10th International Modelica Conference, Lund, Sweden*, Linköping Electronic Conference Proceedings, pages 875–883. Linköping University Electronic Press, 2014.
- Uwe Bau, Heike Schreiber, Franz Lanzerath, and André Bardow. Adsorption-based air-conditioning for battery-driven electric busses. In *24th IIR International Congress of Refrigeration*, 2015.
- Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3), 2003.
- Francesco Bottiglione, Tommaso Contursi, Angelo Gentile, and Giacomo Mantriota. The fuel economy of hybrid buses: The role of ancillaries in real urban driving. *Energies*, 7(7):4202–4220, 2014. ISSN 1996-1073. doi:10.3390/en7074202.
- Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Ltd, Chichester, 2001. ISBN 0-471-87339-X.
- Karin Dietl, Stephanie Gallardo Yances, Anna Johnsson, Johan Åkesson, Kilian Link, and Stéphane Velut. Industrial application of optimization with Modelica and Optimica using intelligent Python scripting. In *10th International Modelica Conference, Lund, Sweden*, Linköping Electronic Conference Proceedings, pages 777–786. Linköping University Electronic Press, 2014. doi:10.3384/ECP14096777.
- Manuel Gräber, Kai Kosowski, Christoph Richter, and Wilhelm Tegethoff. Modelling of heat pumps with an object-oriented model library for thermodynamic systems. *Mathematical and Computer Modelling of Dynamical Systems*, 16(3):195–209, 2010. ISSN 1387-3954. doi:10.1080/13873954.2010.506799.
- Manuel Gräber, Christian Kirches, Hans Georg Bock, Johannes P. Schlöder, Wilhelm Tegethoff, and Jürgen Köhler. Determining the optimum cyclic operation of adsorption chillers by a direct method for periodic optimal control. *International Journal of Refrigeration*, 34(4):902–913, 2011. ISSN 0140-7007. doi:10.1016/j.ijrefrig.2010.12.021.
- Maria Henningsson, Johan Åkesson, and Hubertus Tummescheit. An FMI-Based Tool for Robust Design of Dynamical Systems. In *10th International Modelica Conference, Lund, Sweden*, Linköping Electronic Conference Proceedings, pages 35–42. Linköping University Electronic Press, 2014. doi:10.3384/ECP1409635.
- Dylan Francis Jones, Seyed Keyvan Mirrazavi, and Mehrdad Tamiz. Multi-objective meta-heuristics: An overview of the current state-of-the-art. *European Journal of Operational Research*, 137(1):1–9, 2002. ISSN 03772217. doi:10.1016/S0377-2217(01)00123-0.

Abdullah Konak, David W. Coit, and Alice E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006. ISSN 09518320. doi:10.1016/j.res.2005.11.018.

Daniel B. Leineweber, Irene Bauer, Hans Georg Bock, and Johannes P. Schlöder. An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization (Parts I and II). *Computers & Chemical Engineering*, 27(2):157–174, 2003.

Alexandra Lind, Elin Sällberg, Stephanie Velut, Stephanie Gallardo Yances, Johan Åkesson, and Kilian Link. Start-up Optimization of a Combined Cycle Power Plant. In *9th International Modelica Conference, Munich, Germany*, Linköping Electronic Conference Proceedings, pages 619–630. Linköping University Electronic Press, 2012. doi:10.3384/ecp12076619.

Neal A. Pennington. Humidity changer for air-conditioning: US Patent 2,700,537, 1955. URL <http://www.google.com/patents/US2700537>.

Andreas Pfeiffer. Optimization Library for Interactive Multi-Criteria Optimization Tasks. In *9th International Modelica Conference, Munich, Germany*, Linköping Electronic Conference Proceedings, pages 669–680. Linköping University Electronic Press, 2012. doi:10.3384/ecp12076669.

Spheros GmbH. REVO-E Technical Specifications, 2015. URL <http://www.spheros.de/Produkte/Klimaanlagen/Busse-ueber-12m/REVO-E.html>.

Hubert Thieriot, Maroun Nemer, Mohsen Torabzadeh-Tari, Peter Fritzson, Rajiv Singh, and John Kocherry. Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms. In *8th International Modelica Conference, Dresden, Germany*, Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2011.

Bogdan Tomoiagă, Mircea Chindriș, Andreas Sumper, Antoni Sudria-Andreu, and Roberto Villafafila-Robles. Pareto Optimal Reconfiguration of Power Distribution Systems Using a Genetic Algorithm Based on NSGA-II. *Energies*, 6(3):1439–1455, 2013. ISSN 1996-1073. doi:10.3390/en6031439.

Michael Wetter. Design Optimization with GenOpt. *Building Energy Simulation*, (21):19–28, 2000.

A new Modelica Electric and Hybrid Power Trains library

Massimo Ceraolo¹

¹DESTEC Department, University of Pisa, Italy, massimo.ceraolo@unipi.it

Abstract

This paper describes a new library proposed for simulation of electric and drive vehicle power trains.

Since is a “first approach” library, it does not make usage of the Vehicle Interface Library. It does not overlap with that library, except for minor parts, since the proposed models of electric drives and battery are much more detailed than the simple examples available in the Vehicle Interfaces Library.

This library is fully compatible with both Dymola 2015 and OpenModelica 1.9.2. It is available under the Modelica License 2, and presented at the 11th Modelica International Conference

Keywords: Power Train, Electric drive, Model, Synchronous machine, Asynchronous machine, map-based model, Electric Vehicle, Hybrid Vehicle, Power-split device.

1 Introduction

This paper shows a new small modelica library that is devoted to simulation of vehicular electric power trains.

There are some important reasons to have a specific library for this purpose.

Electric propulsion of vehicles is very important nowadays, since it involves both pure electric vehicles and electric-hybrid vehicles. Its simulation requires somewhat specific models: in fact, detailed models of electrical machines require the variables to follow their sinusoidal variation, that can have frequencies of hundreds or thousands of hertz. Detailed simulation of power converters is even more demanding, since commutation frequencies are easily up to 50 kHz, and therefore time steps must be as small as a few microseconds. This is overkill for vehicular simulations that typically simulate trips lasting several minutes. For instance, the well-known standard vehicle simulation cycle, the NEDC, lasts 1200 s. As a consequence of this, to simulate vehicular propulsion, averaged models must be used, that are sufficiently precise and yet not too demanding in terms of simulation resources.

The library therefore supplies models of electric drives that are a compromise between detail and resource requirements that is adequate for electric vehicles. In addition it supplies other support models that are very important in vehicles, i.e. a battery model and a model for the vehicle drag force.

The library contains four main folders: *MapBased*, *ElectricDrives*, *SupportModels*, *Icons*, *FullVehicles*.

ElectricDrives contains models for synchronous and asynchronous electric drives and simple examples showing their behaviour and how they compare to more detailed MSL models

MapBased. For even more simplified simulations, map-based models of electric drives are satisfactory. Here some models of this kind are proposed, that are tested and then used in some of the full vehicles examples. This folder contains also two ECUs (Electronic Control Units) that are used to control the map-based full vehicle examples.

SupportModels. This folder contains useful models needed, in addition to electric drives, to create full vehicle models: two battery, a drag force and a driver model. The subfolder “Internal” contains models that are internally used to interface mechanical and electrical parts of models, and are not intended for final user usage.

FullVehicles. This folder contains examples of full vehicle power trains built using the supplied models: two electric and to hybrid vehicle models are present.

All the models of this library have been checked and work well with Dymola 2015 and OpenModelica 1.9.2.

The remainder of this paper illustrates the supplied models as well as the full vehicles examples

2 Electric Drives folder

This folder contains models of electric drives based on asynchronous and synchronous machines.

2.1 Asynchronous machine models (AMDrive and AMDrivePU)

The asynchronous machine models are based on Modelica.Electrical.QuasiStationary library. This way, the only dynamics of the machine and drive come from the rotating parts: indeed electrical time constants are typically much lower than those due to mechanical inertias.

Moreover these models offer the advantage of easy interpretation, since the graphical representations reproduce closely the quasi-static single-phase models of electrical machines commonly found in textbooks.

Although the asynchronous machine models provided here are not adequate to simulate machine behaviour unbalanced currents and heavy transients, they cover very satisfactorily the simulations commonly found in electric and hybrid vehicle power trains.

The core of these models is as depicted in figure 1.

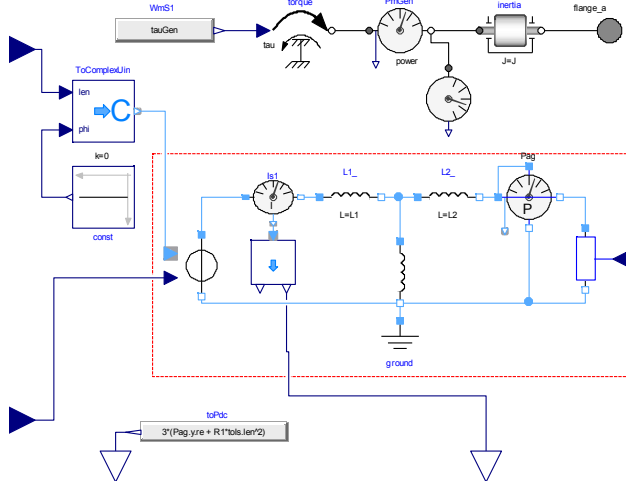


Fig. 1: Representation of the QSAsma model inside the ElectricDrives folder.

The signals entering the model, left in the figure, are the voltage behind stator resistance (rms per phase) and frequency. These are applied to the classical quasi-stationary single-phase asynchronous machine equivalent circuit, where air-gap power and thus torque are evaluated. The generated torque is then applied to the machine inertia (upper part of the picture).

Although it is possible to use this model as is, it was built to be used inside other library models, i.e. AMDrive and its per-unit version AMDrivePU.

The AMDrive arrangement is shown in figure 2.

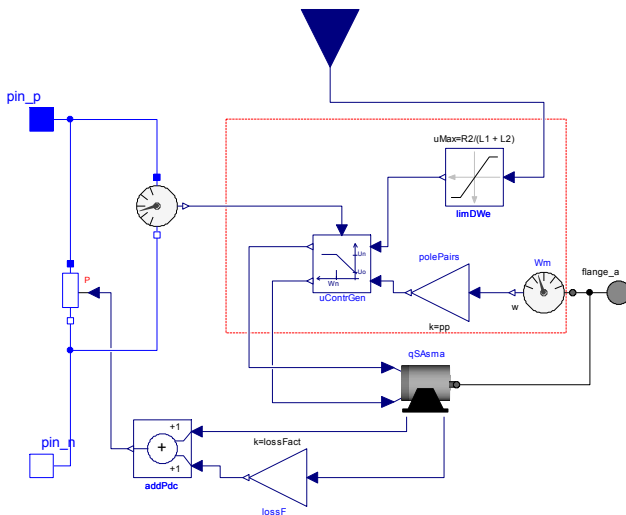


Fig. 2: Internal structure of the proposed WSDrive component.

The previous asma model of figure 1 is used and fed by a control system “uContrGen”, that generates voltage behind stator resistance, according to a classic voltage proportional to frequency rule (Bondea, Nazar, 2006).

Note that the input signal (above in the figure) is the slip frequency dWe , that is intended to be as torque request. In fact, if dWe is not too large, it is nearly-proportional to the asynchronous machine electro-

magnetic torque, as well known from that machine theory. This happens only up to $dWe = R2/(L1+L2)$; that is why the limiter $limDWe$ is introduced in the scheme.

The drive model computes the power absorbed by the drive as the sum, obtained in $addPdc$ model, of $qSAsma$ absorbed power and additional losses created in the $lossF$ block, proportional to machine stator current, that is often a reasonable estimation of inverter losses this because on-state losses often dominate and are proportional to the AC current. The $qSAsma$ absorbed power is computed in $qSAsma$ model as the sum of mechanically generated power and copper losses in $R1$ and $R2$ resistances. However the user can easily enhance this model including additional losses such as iron and stray losses.

A simple test of the asynchronous machine drive is supplied in the library, ElectricDrives. | TestingModels folder, named TestAMDrive.

In this test the proposed drive performance is compared with a similar drive, but obtained based on the MSL standard model of asynchronous machine. This standard model is fed with voltages created using the $U/f = const$ technique similar to that used inside the proposed $qSAsma$, but with quantities that have the actual variable-frequency sine shape.

Just to have an idea of the expected results, in figure 3 the terminal voltages of the two models are compared to each other. Note that $qSAsma$ model gives the rms value; the figure confirms that its behaviour is that of the $aimc$ terminal voltage peaks divided by $\sqrt{2}$.

Similarly, in figure 4 the produced electromagnetic torques are shown.

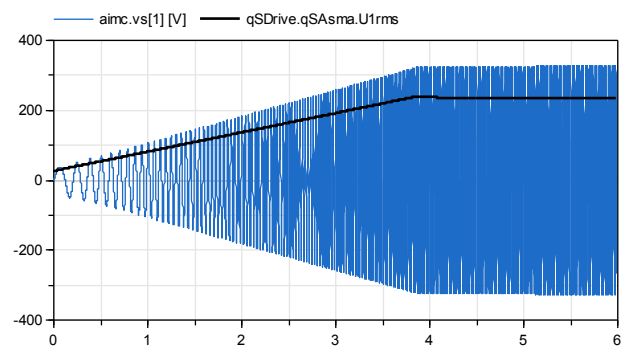


Fig. 3: Comparison of MSL $aimc$ and $qSAsma$ model during a start-up: instantaneous (vs, from MSL), and rms ($U1rms$, from $psDrive$) phase voltage.

A more complex example using the $QsDrive$ model is provided in FullVehicles library folder and described later on in this paper.

A variant of AMDrive called AMDrive PU is also provided. The only difference is that the parameters are expressed in per-unit. This could be useful when the precise machine to be simulated is not known; in this case one could envisage some p.u. values and typically change only, between simulations, the nominal apparent power $Snom$.

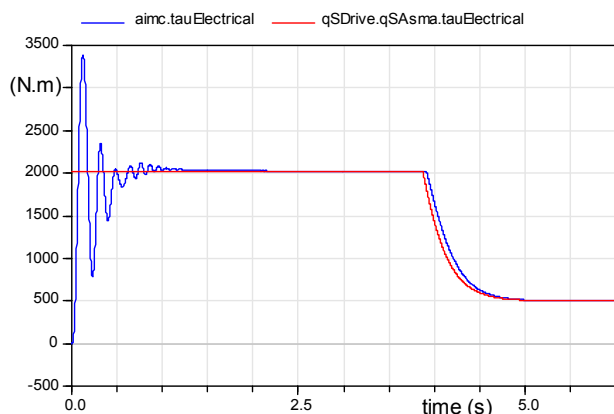


Fig. 4. Comparison of MSL aimc and qSAsma model during a start-up: aimc and qSDrive electromagnetic torques.

2.2 Synchronous, permanent-magnet machine model (PMDrive)

Permanent magnet machines are more and more used on vehicles because of the very advantageous characteristics. Maybe the most important ones are the possibility to have a very large speed range over base speed, and a higher specific power, in comparison to asynchronous machine. The latter feature is mainly due the fact that PMs produce magnetic field without the need of current circulation, differently from the squirrel cage of asynchronous machines. Thus they do not require rotor cooling (Ehsani *et al*, 2006).

The control of PM machines must guarantee that the angle between rotor-generated and stator generated magnetic fields are optimal, i.e. that guarantees the Maximum Torque Per ampere (MTPA) condition. In case of isotropic machine this optimal angle is 90° , while in case of anisotropic one, where the direct-axis reluctance is lower than the quadrature axis one, this optimal value is more than 90 degrees (Schiferl, 1990).

Often instead of the angle between the two fields, the difference between this angle and 90° is considered: this new angle, called gamma-angle, has an optimal value of zero for isotropic machine, while it is larger than zero when $X_q > X_d$. An idea of the optimal gamma-angle trend can be obtained looking at figure 5, in which example combinations of PM flux, and X_d and X_q are considered.

The optimal gamma angle is very easily computed in Modelica: it is just necessary to impose:

$$0 = -\psi_{PM} \sin \gamma + (L_q - L_d) I_s \cos 2\gamma \quad (1)$$

That is just an additional equation to be added to the machine equation set.

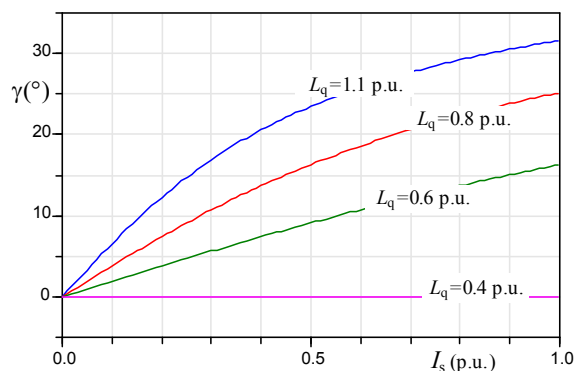


Figure 5. Trend of the optimal gamma angle as a function of stator current for a machine having a magnetic flux $\Psi_{PM}=0.6$ p.u., a direct inductance $L_d=0.34$ p.u., and different values of L_q .

However, as is well known, the optimal gamma can be selected only at low speeds: at very high speeds the flux produced by permanent magnets tends to generate a too high terminal voltage and this tendency must be contrasted by a flux weakening part of the stator current produced field. So the condition determining the gamma angle at these speeds is the one that creates the set voltage at the machine terminals:

$$V_{machine} = V_{set} \quad (2)$$

So equation (1) will be substituted by equation (2) thus retaining the variable-equation balance of the model.

Equations (1) and (2), along which an if-equation to switch between the two, is enclosed in the “AtomicPmsm” model present in the SupportModels folder of the library. This is complemented by a loop that avoids the machine current to overcome a set value in the PmsmAllFluxLimI model, also present in the SupportModels folder. In case of conflicting needs for current, i.e. current is needed to accomplish flux weakening and to produce the requested torque, priority is given to flux weakening.

This way, the usage of PMDrive model, that is rather concise (128 equations) a lot of the PM drive characteristics are provided for:

- automatic selection of optimal gamma angle according to (1) at low speeds
- automatic flux weakening at higher speeds
- automatic switching from optimal angle and flux weakening control
- automatic current limitation
- consideration of machine and inverter losses.

A simple test of the permanent-magnet synchronous machine drive is supplied in the library, ElectricDrives|TestingModels folder, named TestPMDrive.

In this test the proposed drive performance is compared with a similar drive, but obtained based on the MSL standard model of synchronous machine. This standard model is fed with voltages created using pre-

defined Id-Iq pairs, and converting these currents into their time-domain counterpart

Creating a logic that reproduces the optimality obtained in pMDrive just for sake of comparison is too demanding. Therefore in this simulation the values of Id – Iq are simply read in the pMDrive part of the simulation, approximately reproduced by means of two trapezoidal shapes, and then fed into the MSL smpm machine.

Figure 6 shows how the trend of Id (left) and Id (right) are reproduced by trapezoidal sources in the smpm model

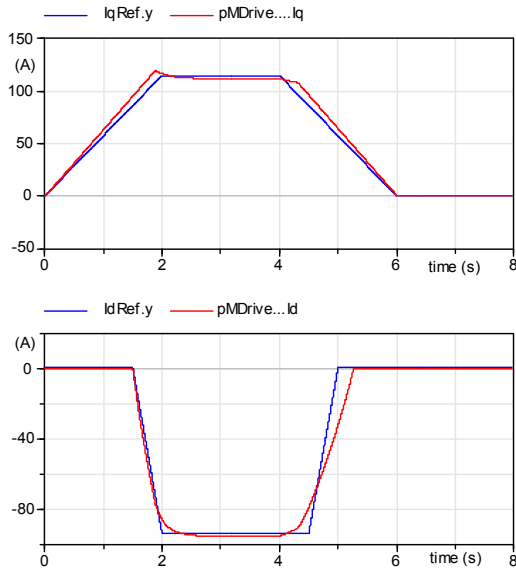


Fig. 6. Trend of Id (left) and Id (right) as reproduced by trapezoidal sources in the smpm model.

Figure 7 shows the voltage on phase 1 as well as the norm of the space-phasor machine voltage that should in principle equal the peak of the sinusoidally varying phase smpm voltages. The red curve shows how the space phasor voltage amplitude $V_{spFF} = \sqrt{V_d^2 + V_q^2}$ could be in case flux weakening would not take place¹.

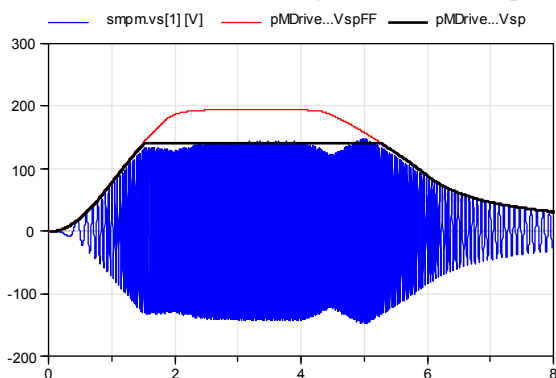


Figure 7. Phase voltage (blue) computed Space Phasor (SP voltage amplitude in absence of flux weakening (red), set SP voltage amplitude (with flux weakening, black).

¹ The two letters “sp” stand for space phasor”, and “FF” stand for “Full-Flux”.

Finally in figure 8 the electromagnetic torque generated in the MSL smpm machine and in the library’s PMDrive are compared.

3 Map-Based folder

In many circumstances the dynamics to be considered in vehicle power train studies are much slower than the faster electric dynamics. In these cases, the only state variables to be considered in electric drive models can be those related with the mechanical inertias of the rotating parts. The rest can be modelled as being algebraic, i.e. with maps containing operating regions and efficiencies.

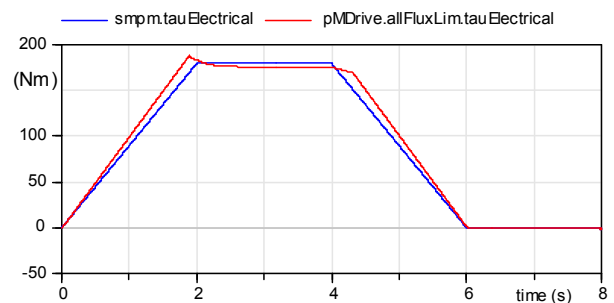


Fig. 8. Comparison of MSL smpm and PMDrive model during a start-up: smpm and PMDrive electromagnetic torques.

To ease simulations in these cases the map-based folder has been provided in the library.

The general arrangement of map-based components can be understood looking at the MBOneFlange model, whose inner architecture is shown in figure 9.

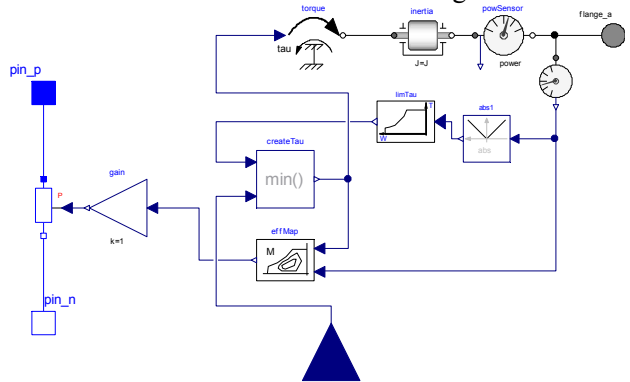


Fig. 9. Internal structure of the MBOneFlange model.

This model models an electric drive, that tries to produce and apply to the inertia the mechanical torque requested from the real input tauRef.

Before applying that torque to the inertia it is verified if the requests is compatible with the drive torque limits as determined by the limTau component, that, as usual in the electric drives, impose a limitation on the delivered torque and delivered power, whatever comes first.

Once the torque is applied to the inertia the applied torque and actual speed determines the operating point of the electric drives. This point is used by the effMap

component that computes the drive losses as a function of the operating point, and requires the absorption from the DC flanges of the total drive power, including losses, by means of a variable resistor, that has as input the power to be drawn.

In a similar way also two –flange electric drive model is internally built.

As regards the ICE model, it follows the same rationale, but instead of efficiency maps it is deemed more natural to use fuel consumption maps The architecture is thus the one shown in figure 10.

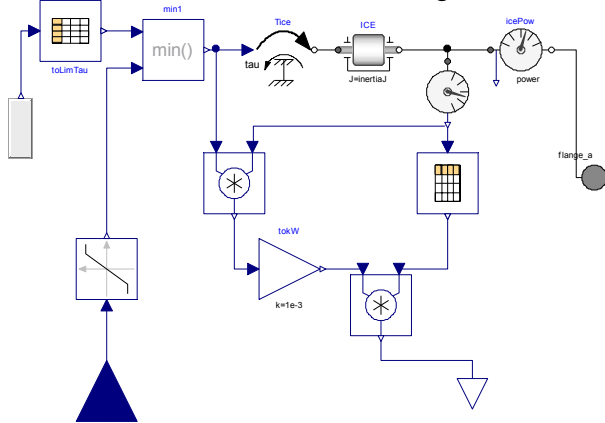


Fig. 10. Internal structure of the MBice model.

The fuel consumption is computed by means of the map “toSpecCons” whose output is in g/kWh, that is multiplied in the block “toG_perHour” times the kW power, to generate the wanted consumption expressed in g/h.

The map-based components come in two flavours: with the set torque being a Real input signal, and with a bus connector.

The components having the bus connector have the advantage that the connectors carry several useful signals, instead of just the set torque. However they have some disadvantages:

- the final user must use the correct names for the bus signals. The usage of connector-based components is recommended, at least at first, either in association with the provided Electronic Control Unit models MBecu1 and MBecu2, or using the converter blocks MBSupport| ToConnIceTauRef and MBSupport| ToConnGenTauRef. Examples of the ECU models usage are provided in FullVehicles folder, while examples of the MBSupport converter blocks are provided in MapBased| TestingModels whose names contain the word “Conn”.
- At most one MBice, one MBOneFlange and one MBTwoFlange components are simultaneously allowed, unless some changes are made on the supplied MB models with bus connectors.

The names that interface with the bus in connector components are those shown in table I. Note that the OneFlangeConn component is called “gen”, while the TwoFlangeConn component is called “mot”. This is a

choice that tends to simplify things, and is consistent with the meaning of the full vehicles models FullVehicles| Psecu1 and FullVehicles|Psecu2.

Table I Names and meaning used as bus signals on the components containing bus connectors.

Name	Sender	Unit	Meaning
iceTauRef	*	Nm	Torque that the ice is requested to deliver
iceW	MBiceConn	rad/s	ICE speed
icePowDel	MBiceConn	W	Power delivered by ICE
genTauRef	MBOneFlangeConn	Nm	Torque that the gen is requested to deliver
genPowDel	MBOneFlangeConn	W	Power that the gen delivers
genTauLim	MBOneFlangeConn	Nm	Maximum torque gen can deliver at the actual speed
motTauRef	MBTwoFlangeConn	Nm	Torque that the mot is requested to deliver
motPowDelA	MBTwoFlangeConn	W	Power that the gen delivers through flange A
motPowDelAB	MBTwoFlangeConn	W	Power that the gen delivers summing flange A and flange B outputs
motTauLim	MBTwoFlangeConn	Nm	Maximum torque gen can deliver at the actual speed

*This is the reference torque that is put on the bus by either ToConnIceTauRef or ToConnGenTauRef or MBecu1 or MBecu2.

All the map-based components are tested in specific testing models.

Here, just as an example, the testMBOneFlange is presented and discussed.

The model is represented in figure 11. It consists on a one-flange drive that is requested to follow a torque profile. The requested torque is larger than that the drive can follow.

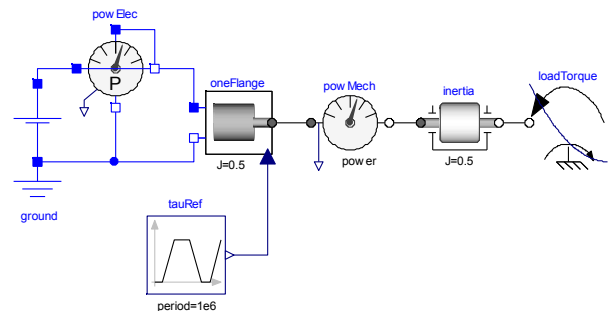


Fig. 11. Test model for the MBOneFlange model.

Some significant plots are shown in figure 12.

It can be noted that:

- during the first 10 seconds the generated torque oneFlange.torque.tau, is 20Nm, as requested from the input. The maximum torque that can be generated is not limited by the power limit (thus state=0)
- between t=10s and 14s the generated torque continues to follow the input signal; but starting from t=10.8s the maximum torque that can be

delivered is limited by the maximum drive power (this is confirmed by the value state=1)

- between t=14 and 18 s, since the drive power has been reached (10 kW), the generated torque is automatically reduced to avoid this limit to be overcome
- between t=18 and t=38 the maximum speed is reached and therefore the generated torque is automatically reduced to avoid this limit to be overcome (state=2)
- above t=38 the torque request is reduced and the drive is again able to deliver this torque.

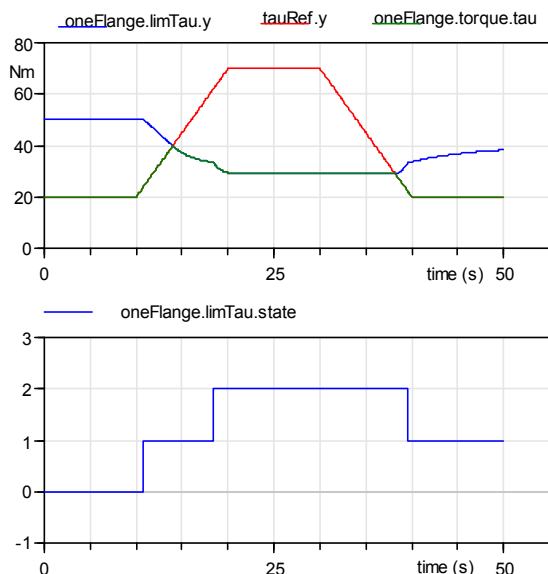


Figure 12. Torques (above) and state (below) of the TestMBOneFlange test model.

Mechanical and electric powers are shown in figure 13. In the central part of the transient, in which speed is constant, the ratio of the two powers represents also the efficiency of the drive in that operating point, that in this case is 85.7%

4 Support models

4.1 PropDriver

Simulation of vehicular power trains normally needs drivers to be simulated.

Indeed in the past also the so-called inverse simulations were common, and used for instance in Advisor software, at least in its royalty-free version distributed by USA’s Department of Energy up to 2003 (EERE 2015). But in recent years direct simulations, typically more realistic and accurate, have become the standard. Direct simulations require the vehicle driver to be simulated along with the vehicle power train.

In the first release EHLibrary, discussed in this paper, a very simple driver model is proposed: the model reads from the hard disk the speed-time profile to be followed, and tries to follow it by means of a purely proportional controller.

Although so simple, it allows very useful simulations to be performed, as demonstrated in the full Vehicle model example provided (section 5).

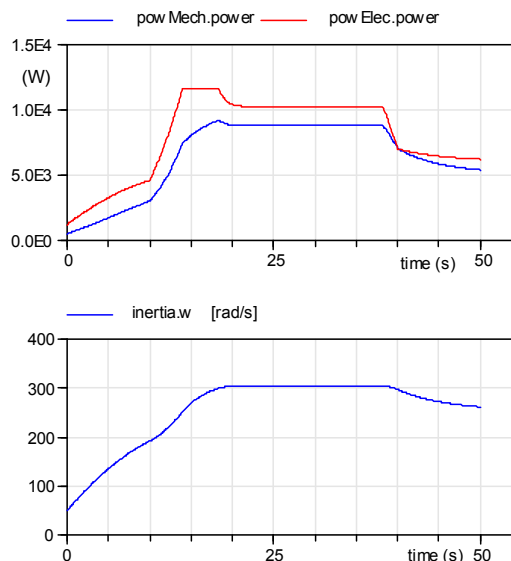


Figure. 13 Mechanical and electrical powers (above) and rotational speed (below) of the TestMBOneFlange test model.

4.2 DragForce

The resistance to movement in vehicles in flat rows is usually expressed by the following well known formula:

$$R = mgf + 0.5 * \rho S C_x V^2 \tag{3}$$

The first term simulates rolling resistances, proportional to vehicle mass m by means of the rolling coefficient f , the second aerodynamic resistance, and therefore is proportional to the air density ρ , front area S , drag coefficient C_x , squared speed.

Indeed formula (3) does not tell all the truth: if we use it in simulations, at zero speed R would be non-zero and the vehicle would start backwards.

To correctly simulate resistance to movement hybrid simulation is needed, e.g.:

$$R = \begin{cases} mgf + 0.5 * \rho S C_x V^2 & \text{if } |V| > 0 \\ F_{est} & \text{if } V = 0 \end{cases}$$

Where F_{est} is the applied external force. The vehicle will start moving whenever F_{est} overcomes mgf .

Fortunately this is easy in Modelica, and the Drag Force component implements this hybrid set of equations. It is directly derived from the MSL component Modelica | Mechanics | Translational | Components | Brake.

4.3 Batt1 and Batt1Conn

A very important piece of hardware for electric and hybrid vehicles is constituted by electrochemical batteries.

The model proposed here derives from the many years of experience of the author in battery modelling, with some useful hints coming from the “BatteryIdealized” model available in the Dassault Systèmes’ SmartElectricDrives library (Dassault Systèmes 2015).

For instance, papers (Ceraolo 2000, Barsali 2002) discuss a whole family of models in which different numbers n of R - C blocks are used to simulate battery dynamics. Although developed for Lead-acid batteries, these models have then proven to be valid for other kinds of batteries, and in particular lithium batteries (Ceraolo 2011) A choice of n corresponds to a given compromise between complexity and precision. In the EHPowerTrain the compromise chosen refers to one R - C block, so that the model can be represented by the equivalent circuit shown in figure 14.

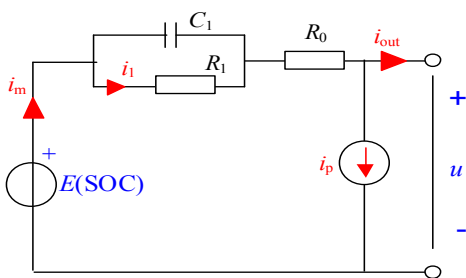


Fig. 14. Battery Equivalent circuit with EMF.

In this model the current that contributes to the charge/discharge process is i_m , while the current flowing in the parasitic branch R_p , i_p is lost. Indeed, the state of charge of the battery can be computed starting from the so called “extracted charge Q_e ” that is the integral of i_m :

$$SOC = 1 - \frac{Q_e}{C_Q} = 1 - \frac{1}{C_Q} \int_0^t i_m(t) dt \quad (4)$$

In (4) it is assumed that when $t=0$ the battery is completely charged, so that Q_e is the charge extracted from the main branch of the electric circuit (the branch in which i_m flows) starting from a fully charged battery.

In general, all the circuit parameters: i_p , R_0 , R_1 , C_1 are function of state of charge and electrolyte temperature; moreover i_p is a non-linear function of the terminal voltage. But for the purpose of EHPowerTrain all these dependences are neglected, except the most important one, i.e. the dependence of E on SOC.

Indeed this dependence in many batteries is nearly linear:

$$E = E_0 + E_1 SOC = E_0 + E_1 \left(1 - \frac{Q_e}{C_Q}\right)$$

Thus in these cases the law relating i_m to E is the same as the one relating current and voltage in a capacitor. In EHPowerTrain linear dependence of E on SOC is assumed, and therefore the circuit of figure 14 is converted into the circuit shown in figure 15.

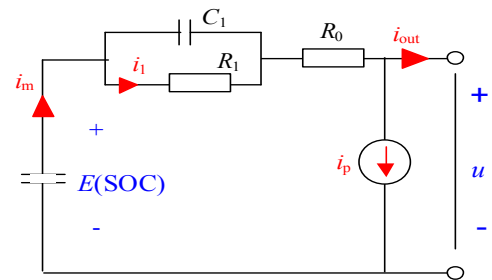


Fig. 15. Battery Equivalent circuit with Capacitor-EMF.

It must be noted that it is expected that the library user knows little about i_p of our model.

Therefore, using a technique that can also be found in SmartElectricDrives Library, i_p is indirectly computed from the full charge/discharge efficiency of the battery. I.e., the user specifies the global constant-current charge/discharge efficiency and from this datum a constant i_p is determined.

Naturally, since i_p cannot be lower than zero and adds loss to the energy loss due to R_0 and R_1 , the user-defined battery efficiency must be not lower than that corresponding to $i_p=0$ condition.

5 Full Vehicles examples

To realistically simulate full vehicles one of the best ways is to use the freely available Vehicle Interfaces library (Modelica Association 2015). However simpler models allow understanding basic things about vehicles more easily. Therefore the EHPowerTrain comes with full vehicle examples that are built from scratch using just modelica and MSL; it is not difficult, however, to include some of the supplied models in the interfaces available in the VehicleInterfaces library, this way taking best of both libraries.

There is one very simplifying assumption that must be noted in the proposed FullVehicles models: the driver outputs a single signal that is a torque reference. When positive it is intended to be traction torque, as could be drawn from the position of the accelerator pedal, while when negative is intended as a brake torque, as could be taken from the brake pedal. This unique torque is sent to the electric or hybrid power trains, that takes care of accelerating or braking. I.e., it is supposed that the power train can perform all the braking actions needed by the speed profile, without additional intervention of mechanical brakes.

These full power train models are stored in the folder named “FullVehicles”. More models are intended to be added in future versions of the library.

In the present version of the library the provided models contain two electric vehicles, one of which (EvAm_bat) based on an asynchronous machine, the other (EvPm_bat) on a permanent-magnet synchronous machine.

The other two (Psecu1 and Psecu2) refer to two hybrid power trains realised with the Power Split

Device component, one of which keeps always on the engine, the other has also some ON/OFF strategy.

Here, for illustration of what these models can do, some results on EvPm_Bat and PSecu2 are proposed.

5.1 EvPm_bat model

This simple vehicle model shows how a PMDrive can be easily and effectively exploited in electric vehicle models. Its appearance is shown in figure 16.

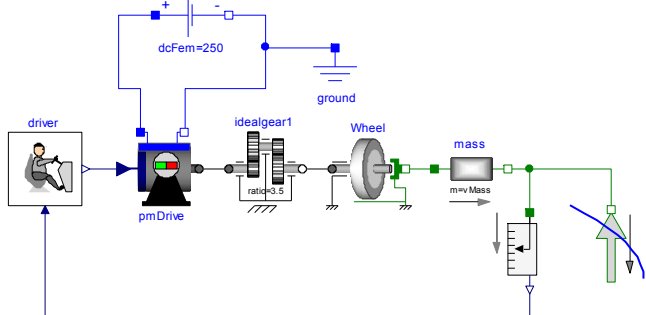


Figure 17: Graphical view of the Evm_bat model.

In this simulation the driver, simulated by means of a simple proportional controller, tends to follow the drive cycle.

The considered drive cycle is the so called "Sort1" standard [UITP 2010]. It is composed by three simple triangles of speed versus time: each of them is composed by three phases: constant acceleration, constant speed, constant deceleration. The maximum speeds of these three triangles are 20 km/h, 30 km/h, 40 km/h respectively.

Figure 18 shows some significant plots.

The proposed simulation refers to a rather small car having a mass of 1300 kg.

In the upper graph the vehicle speed $mass.v$ as long as the wished vehicle speed $driver.from_kmh$. It is seen that this driver is very reactive, and therefore the vehicle speed closely follows the set driving cycle.

In the central plot the generated electromagnetic torque $pmDrive.allFluxlim.tauElectrical$ is shown: the plot indicates that the inertia (acceleration and deceleration) forces dominate. During the constant-speed parts the torque is due only to drag forces, composed of rolling friction and air drag. Since the considered speeds are low, the air drag force is negligible.

Finally, the lower graph shows trend of $I_d(t)$ and $I_q(t)$ that indicate the machine behaviour. Since this machine is anisotropic, torque is determined by I_d and I_q . If an isotropic machine is chosen instead, I_q will determine torque, while I_d will determine terminal voltage; I_d will stay equal to zero at low speeds, to reduce drive current.

Other interesting plots can be drawn from the electrical DC circuit. In the next figure 19, the power delivered by the battery and the State of Charge (SOC) are shown.

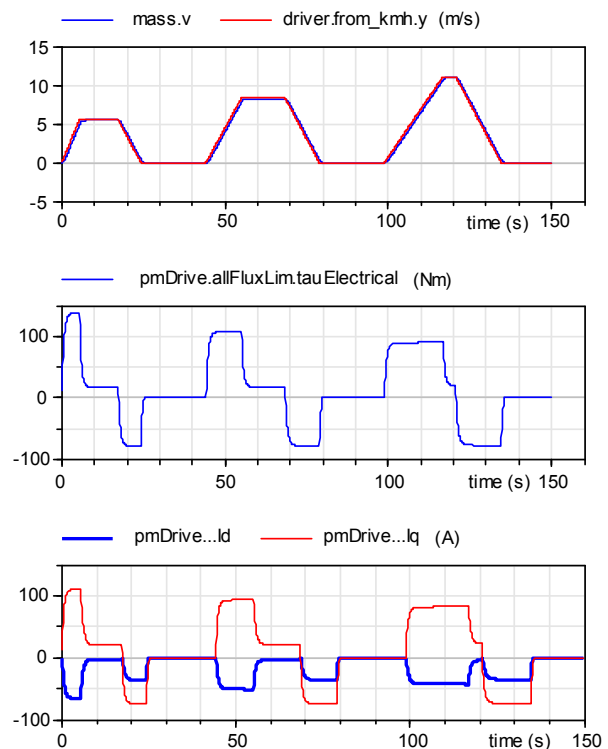


Fig. 18. Some significant plots of EvPm_bat model.

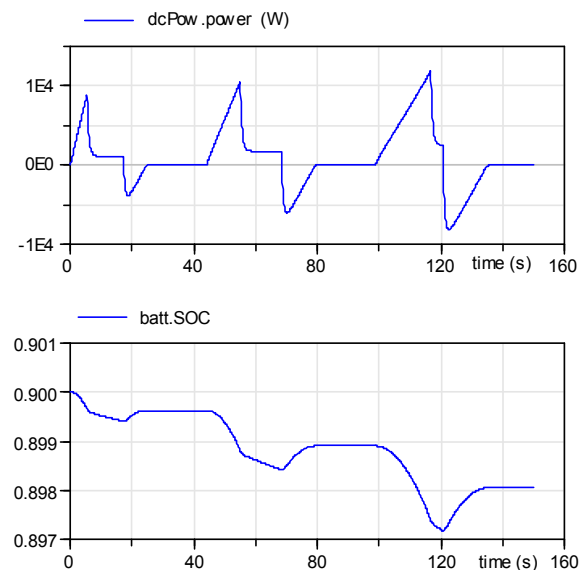


Fig. 19. DC power and SOC of EvPm_bat model.

Note the very small SOC window due to the very short simulation: only three bus stops, for a total of 0.5 km.

The drag force model understands when the vehicle is standstill and switches into locked mode. This can be verified checking as shown in figure 20.

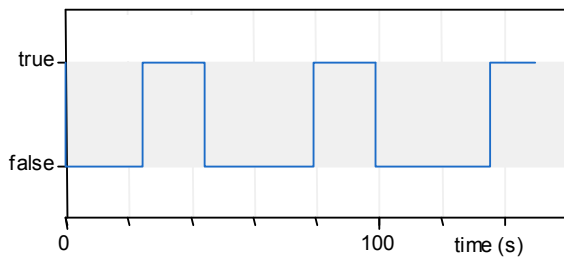


Fig. 20: Value of the boolean variable “dragForce.locked” for the simulation shown in figures 18 and 19.

5.2 PSecu2 model

This model shows a possible behavior of a power train based on a Power Split device (PSD) model. Its appearance is shown in figure 21.

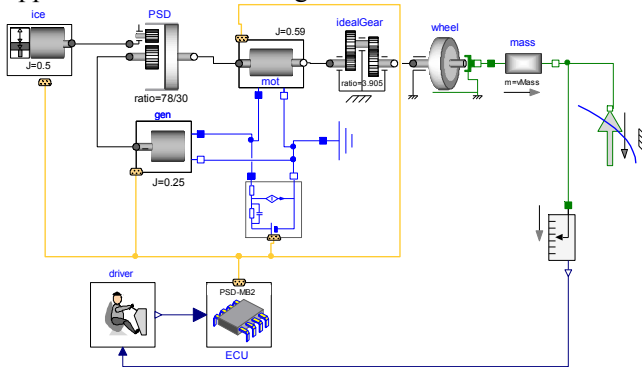


Figure 22: Graphical view of the PSecu2 model.

Moreover it gives the opportunity to show several of the Map-based library models, in the version containing a bus connector. It thus exploits MBOneFlangeConn, MBiceConn, MBTwoFlangeConn components.

The management of all these devices is made in the ECU of the type MBecu2. It tries to satisfy the driver’s torque command in an effective way, considering the ICE consumption fuel map. To do this it has an inner logic that foresees also switching off the ICE at low loads.

The general idea of this logic, that reproduces what is published regarding the first release of Toyota Prius PSD based hybrid, and described in (Toyota, 2003) is as follows:

1. When the ICE is ON it is made operate at its maximum torque, i.e. at the maximum throttle, that for any engine speed corresponds to the lowest specific consumption. Moreover that torque is roughly independent on the ice speed, being always around 90 Nm; this implies that control of the ICE power is obtained just controlling the ICE speed.
2. As a general rule, the ICE speed is chosen to be the value that makes it deliver the load power, measured as an average on the last few minutes of vehicle operation.
3. The one-flange machine, connected to the PSD sun and called “gen”, has as purpose to keep the ICE near its optimal speed.

4. The two-flange machine, connected to the PSD ring through one flange and the final reduction gear through the other, is operated so that the vehicle follows the torque requests from the driver
5. The above logic is modified to keep SOC under control; this control is obtained by action on ICE speed obtained, again, in compliance with the above rule 3, by means of additional action on the gen.
6. Finally, if the engine is delivering for long time too low power, it is temporarily shut down, using a simple ON/OFF technique. Determination of too low ICE power is made by corresponding measure of ICE speed (compare rule 1 above); an hysteresis loop is added to avoid too frequent ON/OFF actions.

When it is decided that switching off the ICE is needed, its reference speed is brought to zero. The given consumption map is such that at zero speed there is also zero consumption. Naturally, although reasonable this is an approximation. In a real case the ICE fuel injection would be brought to zero, and then when also its speed reaches zero some mechanical brake would be activated to allow the PSD to exchange torques with the mechanical objects connected to the other flanges.

In the following figures some results of the simulation proposed in the EHPowerTrain library are proposed.

The considered vehicle is the same as for the EvPm_Bat model. Even the total mass of the two vehicles is taken as equal. However here instead of the Sort1 Cycle, the New European Driving Cycle (NEDC) is used.

Figure 21 shows, from top to bottom, the desired and obtained vehicle speed; the ice rotational speed (when it is brought to zero ICE is set to OFF position and no fuel consumption occurs); the battery SOC that, despite of some fluctuation, shows a general trend to stability due to the specific control loop present in PSecu2.

6 Conclusions

This paper has shown the basic characteristics of a new library, EHPowerTrain, proposed to be presented at the 11th Modelica International Conference.

It is intended for people wanting to simulate electric and hybrid powertrains, with lean models and fast simulations.

The models have thus defined with limited complexity; yet they are able to give interesting results.

All the models work well under Dymola (2015) and OpenModelica (1.9.2).

The library will be made available open-source to the general public, if there is request.

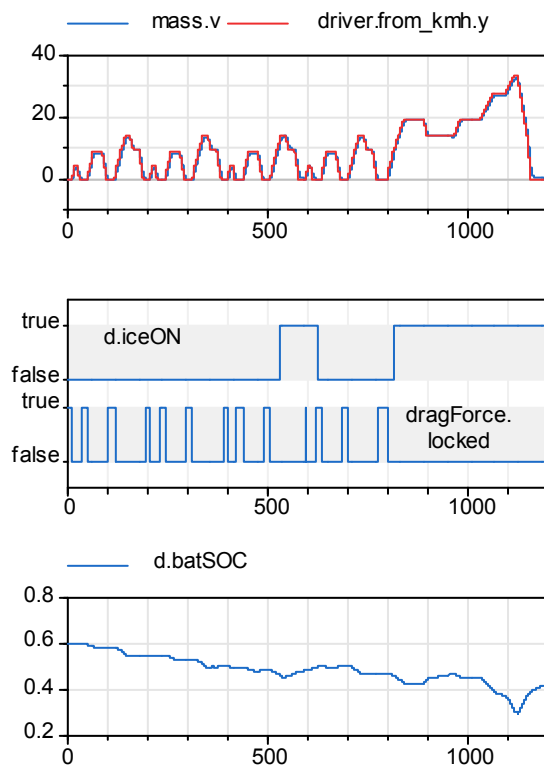


Fig. 23. Some plots related to simulation PSecu2: desired and actual vehicle speed (m/s, top); Ice-ON and vehicle Stop signals (middle), Battery SOC (dimensionless, bottom).

References

- Bondea, S. A. Nazar; "Electric Drives", CRC Press, Taylor & Francis Group, 2006 ISBN 0-8493-5220-1, section 8.17.
- UITP Project SORT: Standardised On-Road Test Cycles, 2010; data for ordering on www.uitp.org;
- M. Ehsani, Y. Gao, A. Emadi: "Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design", CRC Press, 2009, ISBN 9781420053982
- R. Schiferl, T. Lipo: "Power Capability of Salient Pole Permanent Magnet Synchronous Motors in Variable Speed Drive Application", *IEEE Transactions on Industry Applications*, l. 26, N. 1, Jan/Feb 1990
- Toyota documentation <http://www.evworld.com/library/toyotahs2.pdf>, May 2003, retrieved from the Internet on 2015
- M. Ceraolo: "New Dynamical Models of Lead-Acid Batteries", *IEEE Transactions on Power Systems*, November 2000, Vol. 15, N. 4, pp. 1184-1190.
- S. Barsali, M. Ceraolo: "Dynamical models of lead-acid batteries: implementation issues", *IEEE Transactions on Energy Conversion*, Vol. 17, N. 1, Mar 2002, Pages 16-23.
- M. Ceraolo, T. Huria, G. Lutzemberger: "Experimentally determined models for high-power lithium batteries", Book *Advanced battery technology*, ISBN: 978-0-7680-4749-3 doi:10.4271/2011-01-1365. Also presented at the *SAE 2011 World Congress*. Cobo Center Detroit, Michigan (USA), 12-14/4/2011.

Dassault Systèmes Smart Electric Drives Library documentation: <http://www.3ds.com/fileadmin/PRODUCTS/CATIA/DYMOLA/PDF/dymola-smart-electric-drives-library.pdf>; File available for download on April 2015

Modelica Association: <https://www.modelica.org/libraries>, Vehicle Interfaces library Link available on April 2015.

EERE Information Center, https://www1.eere.energy.gov/vehiclesandfuels/pdfs/success/advisor_simulation_tool.pdf file available for download on April 2015.

Initiatives for acausal model connection using FMI in JSAE (Society of Automotive Engineers of Japan)

Yutaka Hirano¹ Satoshi Shimada² Yoichi Teraoka³ Osamu Seya⁴
Yuji Ohsumi⁵ Shintaroh Murakami⁶ Tomohide Hirono⁷ Takayuki Sekisue⁸

¹Toyota Motor Corporation, Japan, yutaka_hirano@mail.toyota.co.jp

²Honda R&D Co., Ltd., Japan, satoshi_shimada@n.t.rd.honda.co.jp

³Mazda Motor Corporation, Japan, teraoka.yo@mazda.co.jp

⁴DENSO CORPORATION, Japan, OSAMU_SEYA@denso.co.jp

⁵AZAPA Co., Ltd., Japan, yuji-ohsumi@azapa.co.jp

⁶Dassault Systèmes K.K., Japan, Shintaroh.MURAKAMI@3ds.com

⁷NewtonWorks Corporation, Japan, hirono.tomohide@newtonworks.co.jp

⁸ANSYS Japan K.K., Japan, takayuki.sekisue@ansys.com

Abstract

Authors initiated trial and evaluation of a new method to connect physical ports of acausal model and causal signal ports using FMI as an activity of technical committee of JSAE (Society of Automotive Engineers of Japan). We propose a way of model export and connection using new adaptor models. This method was tested by a benchmark model of a control system. Simulation results for the benchmark model showed good consistency between the original acausal model and the connected model using FMUs separated from the original model by this method. Also a guide-line about using FMI for the model connection using this method was made in JSAE and is distributed to general users of Japanese automotive industries. Finally expectations about future enhancement of FMI for model exchange and circulation between different companies are presented.

Keywords: FMI, Model Exchange, Acausal Physical Connector

1 Introduction

Importance of utilizing simulation is increasing for the development of automotive systems because both high functionality and high reliability are required while the development time is becoming shorter. For large-scale and multi-domain development of automotive systems, environment for development which enables to connect simulation models developed in various companies is becoming important more and more. Though, in automotive industries in Japan, various kinds of physical modeling tools are used for each physical domain by each organization. Thus it was difficult to connect those models easily.

Upon above background, the Committee on Research of Model Development and Circulation Methods Based on Global Standardized Description was established since March 2012 in JSAE. In the

Committee, Working Group for Model Connection Technologies was started to try and evaluate the efficacy of model connection using FMI. In that activity authors found that there was a problem of algebraic loop generated by diving and connecting sub-models (FMUs) because FMI only provided the way to connect models by causal signal flows, i.e. defining causality of input and output of components is required. On the other hand, acausal modeling tools as Modelica tools can handle the problem of algebraic loop by causality analysis and symbolic manipulation of equations. Thus it seemed effective to connect FMUs in acausal modeling environment by converting causal signal ports to acausal physical port and vice versa. In the following section, the method to use special adaptor models to divide and connect sub-models via FMI is proposed. The method was validated by benchmark models developed in JSAE committee. Comparison between the simulation result of the original acausal model and that of a unified model of divided FMUs was done and we got good consistency of the simulation results. Finally some expectations for future enhancement of FMI for model exchange from Japanese automotive industries are described.

2 Benchmark Model of a Simple Control System

2.1 Structure of the Benchmark Model

Figure 1 shows one benchmark model of JSAE used for the evaluation of model connection methods using FMI. This model is a simple control system of DC motor for rotational angle feedback control as shown in **Figure 2**.

This simple control system consists of three major sub-systems: PI controller, DC motor and rotational mechanical system. In each sub-system, the system of equations are as follows.

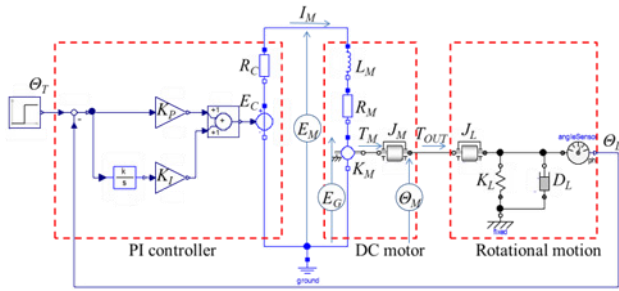


Figure 1. Benchmark model (Simple control system)

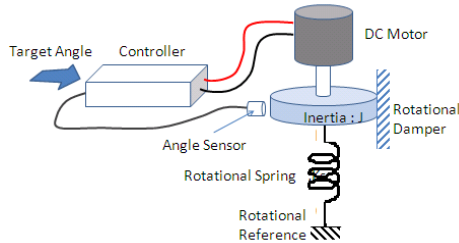


Figure 2. Physical image of the simple control system

PI Controller:

$$E_c(s) = K_p \cdot [\theta_T(s) - \theta_L(s)] + K_i \cdot \frac{1}{s} [\theta_T(s) - \theta_L(s)] \quad (1)$$

$$E_M(s) = E_C(s) - R_C \cdot I_M(s) \quad (2)$$

DC Motor:

$$E_M(s) = L_M \cdot s I_M(s) + R_M \cdot I_M(s) + E_G(s) \quad (3)$$

$$E_G(s) = K_M \cdot s \theta_M(s) \quad (4)$$

$$T_M(s) = K_M \cdot I_M(s) \quad (5)$$

$$T_{OUT}(s) = T_M(s) - J_M \cdot s^2 \theta_M(s) \quad (6)$$

Rotational Mechanical System:

$$T_{OUT}(s) = J_L \cdot s^2 \theta_L(s) + D_L \cdot s \theta_L(s) + K_L \cdot \theta_L(s) \quad (7)$$

$$\theta_M(s) = \theta_L(s) \quad (8)$$

Here,

$E_C(s)$: Voltage of voltage generator in PI controller

$E_M(s)$: Terminal voltage of DC motor

$E_G(s)$: Electromotive voltage of DC motor

$I_M(s)$: Current flow of DC motor

$T_M(s)$: Driving torque of DC motor

$T_{OUT}(s)$: Output torque of DC motor

$\theta_T(s)$: Target rotational angle

$\theta_M(s)$: Rotational angle of DC motor

$\theta_L(s)$: Rotational angle of the mechanical system

and

K_p : Proportional gain of PI controller

K_i : Integration gain of PI controller

R_C : Internal resistance of PI controller

L_M : Inductance of DC motor

R_M : Resistance of DC motor

K_M : Current - torque coefficient of DC motor

J_M : Inertia of DC motor

J_L : Inertia of the rotational mechanical system

D_L : Damping coefficient of the mechanical system

K_L : Spring coefficient of the mechanical system

2.2 Adapter Model

Let's consider to make FMU from the DC motor sub-model of Figure 1. Here, one portion to divide the model is the electronic connector between the electronic output of PI controller and input of the DC motor. Also the mechanical connector between the mechanical output of DC motor and the mechanical input of the rotational motion system should be chosen as dividing portion. It is important to choose the appropriate connectors which coincide with the actual interconnection of parts and systems to handle models provided by different suppliers easily.

In general, it is the most convenient that such models are described by acausal modeling tool such as Modelica tools because there is no necessity to consider the causality of each system of equations when assembling the models. But it is still not realized that every model of necessary sub-systems are made by acausal modeling tools. There still are many existing models made by different tools supporting only causal modeling.

Thus, it is important to enable those causal models to be connected by using FMI in acausal modeling environment. However, because FMI is based on only causal signal connection, it is necessary to prepare adaptor models to translate acausal physical port to causal signal connectors. **Figure 3** shows the proposed models of adaptors for electric, rotational mechanical and translational mechanical domains.

In electronics adaptors shown in **Figure 3**, the connectors shown by rectangular terminal are acausal physical ports. The connectors shown by triangle terminal are causal signal connectors of voltage and current, and the direction of the triangle head shows the direction of the corresponding signal flow. There are following equations between each variable.

$$E1_{pin} = v1 \quad (9)$$

$$i1 = I1_{pin} \quad (10)$$

$$v2 = E2_{pin} \quad (11)$$

$$I2_{pin} = -i2 \quad (12)$$

In rotational mechanical adaptors shown in **Figure 3**, the connectors shown by circle terminal are acausal physical flanges. The connectors shown by triangle terminal are causal connectors expressing physical signals of rotational angle, velocity, acceleration and torque. As same as the electronic adaptors, the direction of the triangle head shows the direction of the corresponding signal flow. There are following equations between each variable.

$$\theta1_{flange} = \theta1 \quad (13)$$

$$\Omega1_{flange} = \omega1 \quad (14)$$

$$A1_{flange} = a1 \quad (15)$$

$$\tau1 = T1_{flange} \quad (16)$$

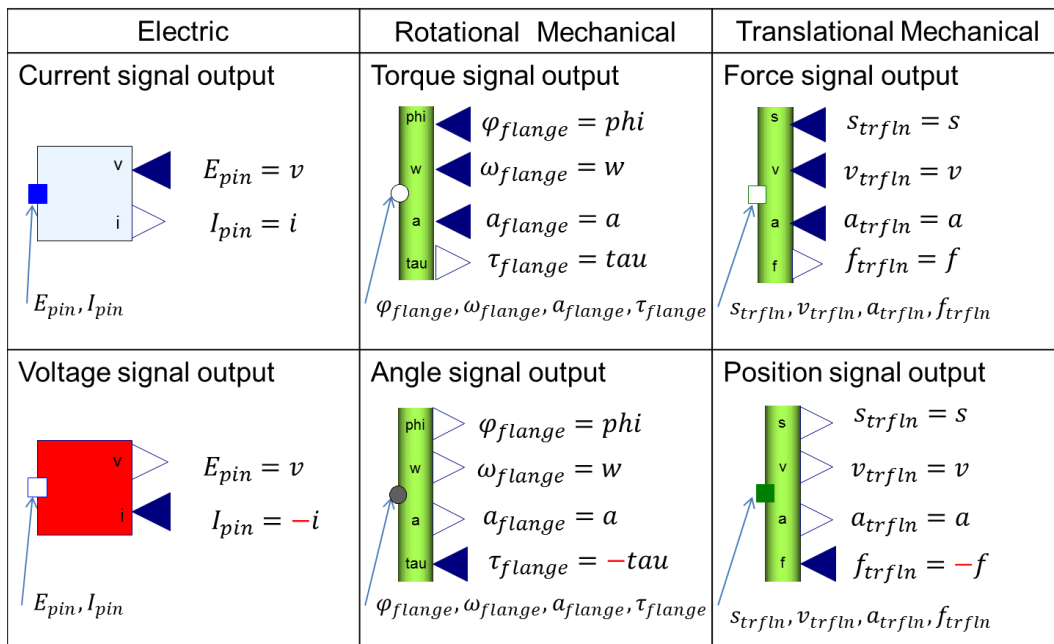


Figure 3. Adapter models to connect acausal physical port and causal signal ports

$$\theta_2 = \theta_{2flange} \quad (17)$$

$$\omega_2 = \omega_{2flange} \quad (18)$$

$$a_2 = a_{2flange} \quad (19)$$

$$T_{2flange} = -\tau_2 \quad (20)$$

There is similar relationship between the signals of acausal physical port and causal signal connectors also for the mechanical translational adaptors.

It is important to notice that there are minus signs in the equation (12) and the equation (20). To decide the sign of every flow variables of acausal connector by integrated way, we define the polarity of flow variables as follows.

- **Sign of flow variables defined as output of causal connector is positive when the flow goes out from the component.**
- **Sign of flow variables defined as input of causal connector is positive when the flow comes into the component.**

By above definition, the function of the adaptor models should become like bellows.

- When transferring flow variable(s) coming into the component at acausal connector to that of causal connector, the sign of the variable(s) is plus. (It's not necessary to change the polarity of both signals as shown in the equation (10) and (16).)
- When transferring flow variable(s) going out of the component at causal connector to that of acausal connector, the sign of the variable(s) is minus because of the definition of Modelica Standard Libraries (MSL): i.e. the sign of flow variables is plus when they come into the component. This

means that it is necessary to invert the sign of the corresponding flow variables as shown in the equation (12) and (20).

By above definition of signal flows, it becomes possible to utilize the functionality of Modelica tools to generate the equation about flow variables to be summed to zero when the physical connectors are connected. This feature enables us to connect FMUs generated by using proposed adaptors into acausal modeling environment. By this way, it becomes possible to utilize the functionality of Modelica translator to handle algebraic loops for connected model of multiple FMUs. We have proposed the above definition of the adaptor models to Modelica Association in March 2014. It is desired that this kind of adaptor models will be prepared in coming future, hopefully as a part of MSL, also for other physical domains such as heat transition system, liquid system and so on.

2.3 Generation of FMU

Next we will show the way to split the sub-model of DC motor by using adaptor models mentioned in the above section. We can make the sub-model as shown in Figure 4 by using one mechanical adaptor and two electronic adaptors.

Equations for the DC motor become as follows.

$$i_{M2_OUT}(s) = \frac{v_{M1_IN}(s) - v_{M2_IN}(s) - E_G(s)}{L_M \cdot s + R_M} \quad (21)$$

$$i_{M1_OUT}(s) = -i_{M2_OUT}(s) \quad (22)$$

$$E_G(s) = K_M \cdot \omega_{M_IN}(s) \quad (23)$$

$$T_M(s) = K_M \cdot i_{M2_OUT}(s) \quad (24)$$

$$\tau_{M_OUT}(s) = T_M(s) - J_M \cdot a_{M_IN}(s) \quad (25)$$

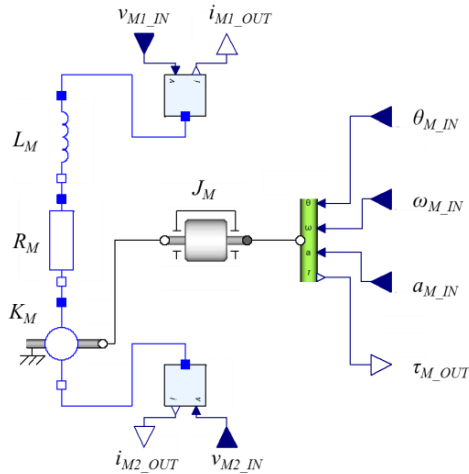


Figure 4. DC motor model with adaptors

Now we can generate FMU from the sub-model of the DC motor shown in Figure 4 because all of the variables which are used for interconnecting to other models are defined by causal (i.e. defined as either of input or output) connectors.

2.4 Importing and Connection of generated FMU

When importing the generated FMU in the host tool (i.e. by Import for ModelExchange or Master for Cosimulation), the imported FMU looks like Figure 5. (The appearance is different according to the used tool.)

Inside the imported FMU model, the relationship between the input and output becomes as following equations (26) to (28).

$$i_{M2_OUT} \tag{26}$$

$$= f_1(v_{M1_IN}, v_{M2_IN}, \theta_{M_IN}, \omega_{M_IN}, a_{M_IN}) \tag{27}$$

$$i_{M1_OUT} = -i_{M2_OUT} \tag{27}$$

$$\tau_{M_OUT} = f_2(v_{M1_IN}, v_{M2_IN}, \theta_{M_IN}, \omega_{M_IN}, a_{M_IN}) \tag{28}$$

The definition of each function f_1 and f_2 cannot be seen from outside. The actual equation of functions are implemented as dll file in the fmu file (in the case of using Windows OS).

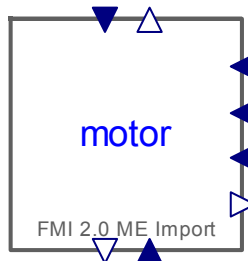


Figure 5. Imported FMU model of DC motor

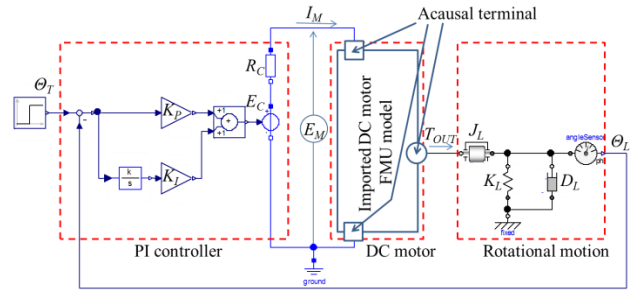


Figure 6. System model connecting imported DC motor FMU model

Next, we will connect the imported FMU model of the DC motor within the total system model as shown in Figure 6. In this model, the acausal sub-model of DC motor in Figure 1 is just replaced by the imported FMU model. The problem is how to connect the causal FMU model to the acausal physical model of the total system. We can solve this problem also by using adaptor models mentioned above. In the case that the FMU model outputs voltage signal and inputs current signal, then we should connect the adaptor which connects electronic acausal port with voltage signal as input and current signal as output shown in Figure 3. Similarly, if a terminal of rotational mechanics outputs torque signal and inputs signals of rotational angle, velocity and acceleration, then we should connect the adaptor which connects mechanical acausal connector with torque signal as input and signals of rotational angle, velocity and acceleration as output shown in Figure 3. In the case of FMU model shown in Figure 5, the necessary model connected with adaptor models becomes like Figure 7.

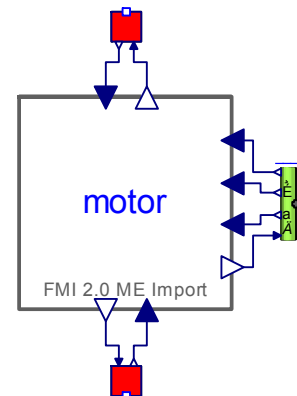


Figure 7. FMU model with adaptors

Now we can connect the FMU model with adaptors into the acausal total model as shown in Figure 6. Figure 8 shows the result of connecting the FMU model of the DC motor using the adaptors.

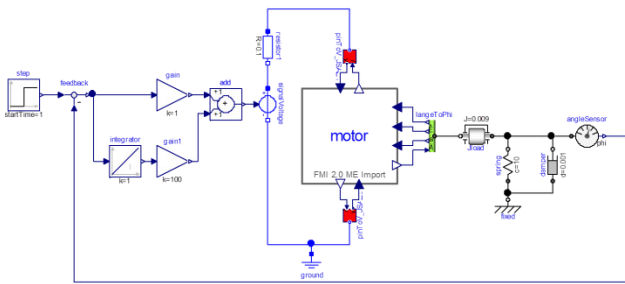


Figure 8. System model with DC motor FMU model

Similarly, separated FMU models of PI controller and rotational motion system can be generated by using proper adaptor models as shown in Figure 9. Then generated and imported FMU models of those sub-systems become as shown in Figure 10. Finally we can make a total system model using those three FMUs as shown in Figure 11. In this example, FMU of both PI controller and DC motor output current signal. Also FMU of both DC motor and rotational motion system output torque signal. Please be aware that by using the adaptors it becomes possible to connect output signals each other between two FMUs via acausal connection because causality is solved by Modelica translator. It becomes error if the causal output signals of FMU are directly connected without using adaptors.

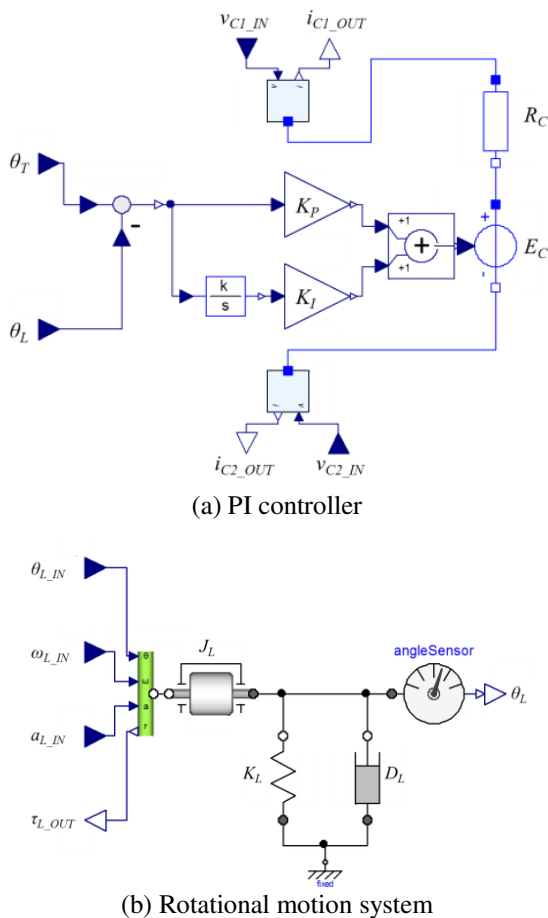


Figure 9. Separated models for FMU

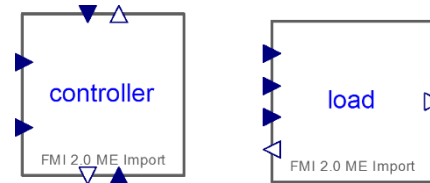


Figure 10. Imported FMU models

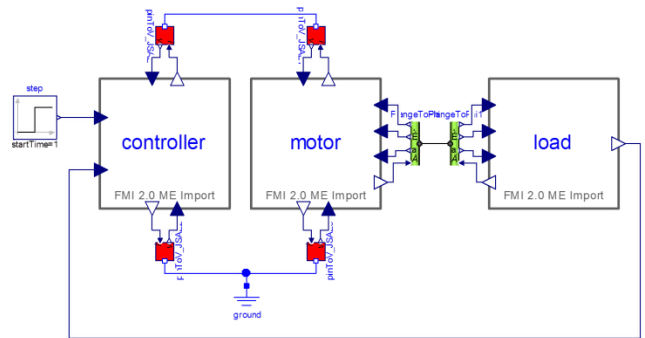


Figure 11. System model using 3 FMUs

2.5 Simulation Results

Figure 12 shows simulation results of the rotational angle of load shaft for the benchmark model. The results of the original model without using FMU (Figure 1), the model using only FMU of DC motor (Figure 8) and the model using all three FMUs (Figure 11) are compared. It is confirmed that all the results are identical. Here, the parameter of the models are set so that the results become oscillatory and the comparison is easy. All FMUs were made by ModelExchange mode.

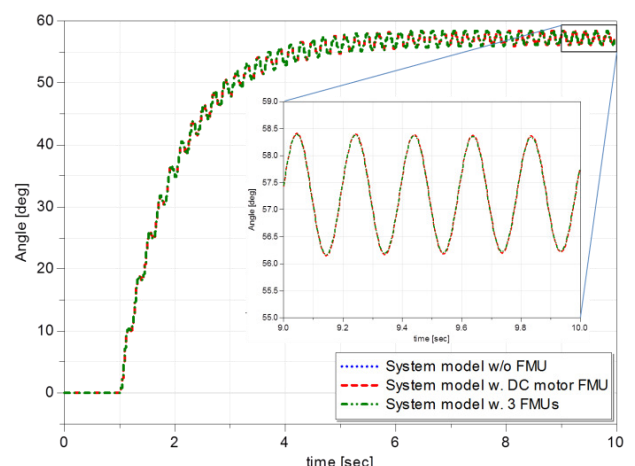


Figure 12. Simulation results (Rotation angle of load shaft)

3 Full Vehicle Simulation Using FMUs from Different Tools

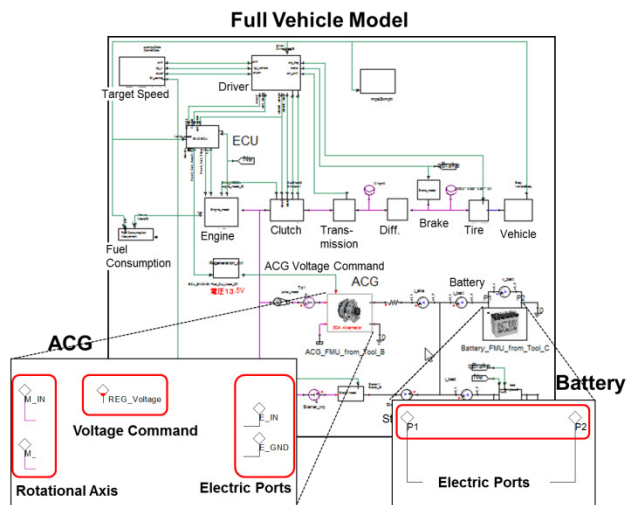


Figure 13. Full vehicle model for electric system evaluation

Another test model to evaluate the electric system using full vehicle model was developed and tested. Figure 13 shows the total model of the test model generated by one VHDL-AMS tool (Sekisue et al., 2013). Here, we tried to replace sub-models of AC generator (ACG) and battery with FMUs generated by other Modelica tools. As same as the benchmark model shown in Figure 1, we introduced adaptor models to connect acausal physical terminals and signals of causal connectors.

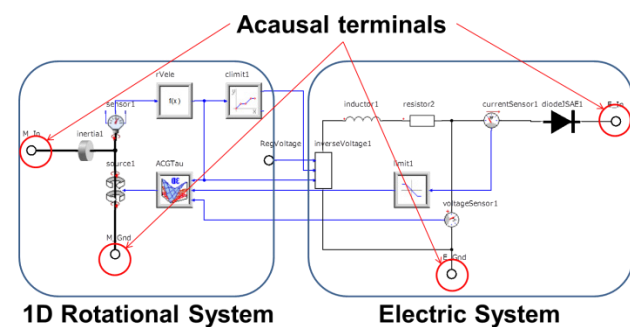


Figure 14. Model of ACG

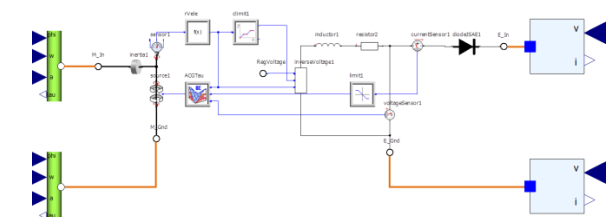


Figure 15. Model of ACG with adaptor models

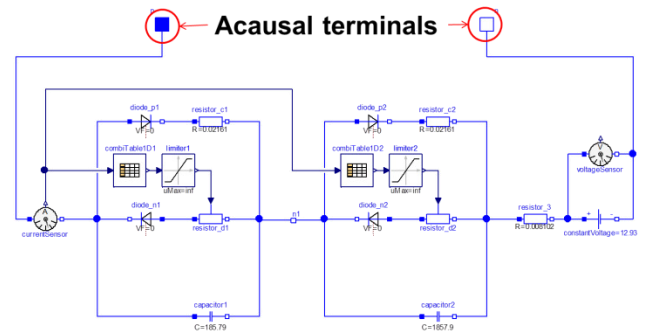


Figure 16. Model of battery

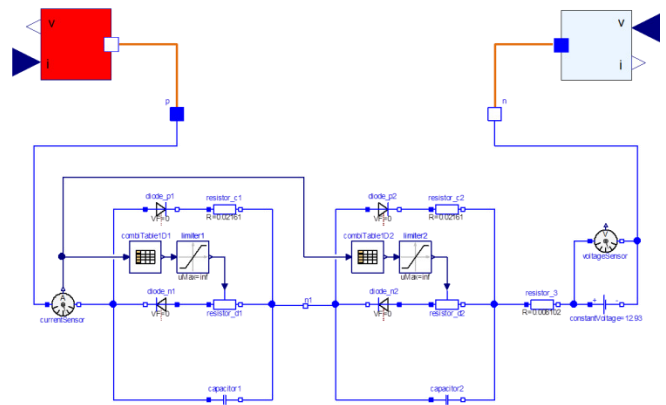


Figure 17. Model of battery with adaptor models

Figure 14 shows the model of ACG generated by one Modelica tool. The model to be converted to FMU was generated as shown in Figure 15 by adding the adaptor models. On the other hand, battery model was made using another Modelica tool as shown in Figure 16. Similarly, the modified model to be converted to FMU was made as shown in Figure 17. It was necessary to make the above-mentioned adaptor models also in the VHDL-AMS tool and connect FMUs from each Modelica tool as shown in Figure 18.

Finally simulation test was done in the VHDL-AMS tool. One example of the results is shown in **Figure 19**. It was confirmed that model connection using FMUs from different tools was successful by using the proposed adaptor models.

4 Summary and Future Requests

We proposed and tested a method using some kind of adaptor models for FMU to realize following functions.

1. Generating FMUs which have connectors of causal signals from acausal modeling tool.
2. Converting causal FMUs to acausal sub-models and connecting them by acausal modeling way.

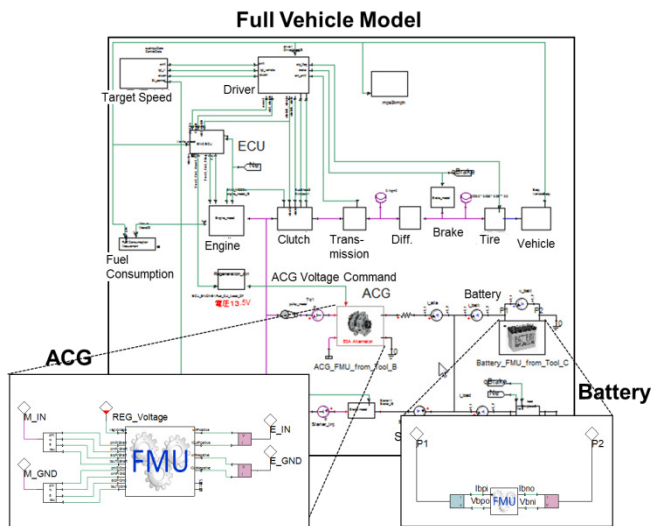


Figure 18. Full vehicle model using FMUs with adaptor models

This technique enabled us to connect sub-models developed by different tools (for example, Modelica tool and VHDL-AMS tool), though it is necessary to make adaptor models in each tool. Additionally we made a guideline for using this technique and published in the web page of JSAE (JSAE Committee, 2014). (Currently only available in Japanese.)

As future works, we plan to do following tasks.

- Extend this method also for FMI for CoSimulation.

- Enhance activity for model development and exchange between automotive industries using FMI.
- Push tool vendors to support the newest version of FMI.
- Request to make better specification of FMI for actual usage.

As for the last activity, there is high expectation to the activity by FMI Working Group of Modelic Association to realize that future FMI will support automatic decision of causality of signals (i.e. the definition of input and output). Currently user should decide the causality of signals of FMU so that there is no conflict when combining multiple models, but this task is very troublesome. Additionally it is desired that as much as possible tools will support automatic handling of algebraic loops generated by connecting multiple FMUs by utilizing above capability.

References

JSAE Committee on Research of Model Development and Circulation Methods Based on Global Standardized Description, Guideline of Model Connection using FMI in Acausal Modeling Tools, 2014 (Available online at <http://www.jsae.or.jp/tops/topics/1241/1241-1A.pdf>).

T. Sekisue, K. Tsuji, M. Ogawa, T. Fukada, K. Tanimoto, S. Hikida, M. Ueda, T. Kato, Alternator model for full vehicle simulation, Proc. *JSAE annual conference 2013 Spring*, No.407-20135490, 2013 (in Japanese).

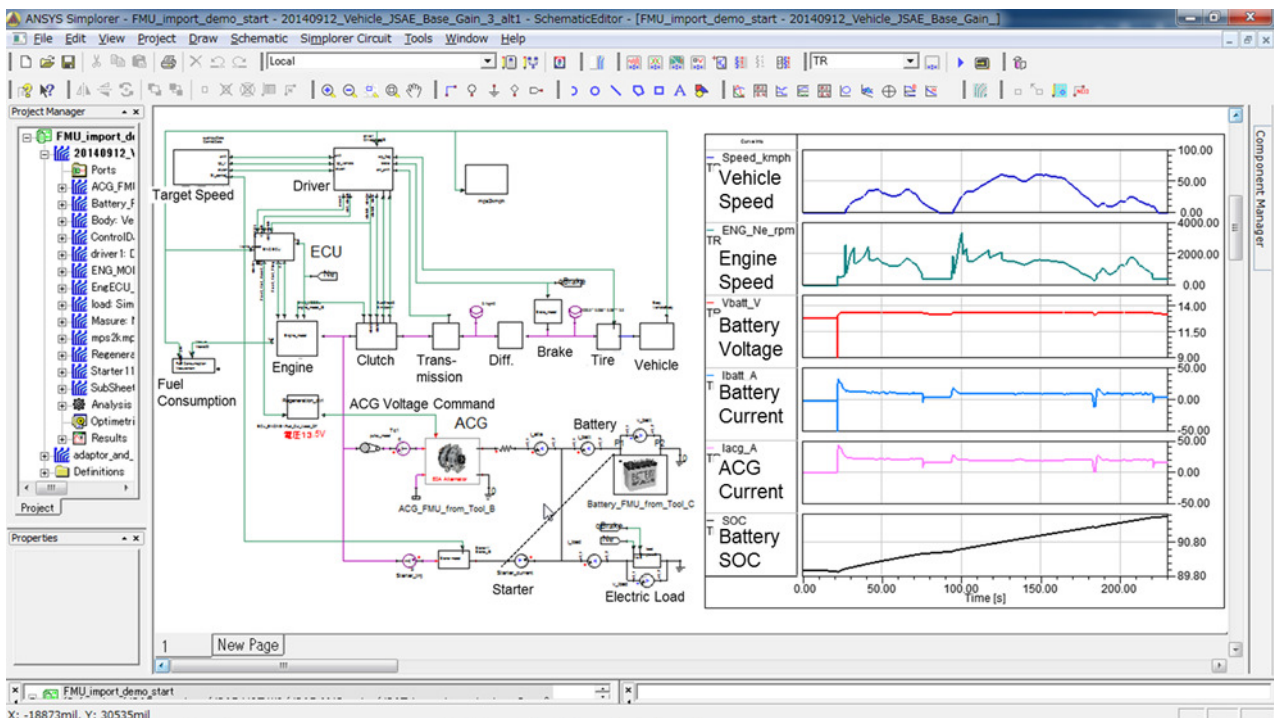


Figure 19. Simulation result of the full vehicle model

Dynamical Model of a Vehicle with Omni Wheels: Improved and Generalized Contact Tracking Algorithm

Ivan Kosenko¹ Sergey Stepanov² Kirill Gerasimov³ Alexey Rachkov⁴

¹Department of Theoretical Mechanics, Moscow Aviation Institute, Russia, kosenko@ccas.ru

²Department of Mechanics, Dorodnitsyn Computing Center of RAS, Russia, stepsj@ccas.ru

³Department of Theoretical Mechanics and Mechatronics, Lomonosov Moscow State University, Russia, kiriger@gmail.com

⁴Department of Theoretical Mechanics, Moscow Aviation Institute, Russia, alexey-rachkov@yandex.ru

Abstract

A model of the multibody dynamics for an omni wheel assuming embedded in a frame of wider dynamical environment of the whole vehicle is under development and verification. Modelica base classes developed earlier for the multibody applications with contacts involving friction are used. Generalization has been performed for the model of contact tracking algorithm between roller and horizontal floor. Generalization includes non-zero angle between the roller axis of rotation and plane of the omni wheel. Contact tracking algorithm is implemented in two cases: (a) implicit and (b) explicit.

Models for these cases (a) and (b) are currently “embedded” into the omni vehicle model earlier verified. For simplicity we analyze a multibody system comprising the wheel plus set of rollers being mounted along its circumference. A remainder of the vehicle is replaced by the wrench properly arranged in a way such that the wheel keeps its vertical orientation permanently. The performed computations have shown that two algorithms of the contact tracking generate completely identical dynamics of the whole multibody system.

Keywords: omni wheel; contact tracking; unilateral constraint; angled rollers; model of friction

1 Introduction

A construct of the omni vehicle (Ilon, 1975) dynamical model has been presented in (Kosenko and Gerasimov, 2014), see also papers (Kálmán, 2013; Tobolár et al., 2009). Simplified model for roller mounting on the wheel disk has been considered there: the roller axis of rotation assumed to be in the disk, or equivalently the angle between this axis and the wheel plane, denote it by ψ , is equal to zero. We will call this angular parameter of the model the angle of the preliminary roller rotation (pre-rotation) about the wheel radius intersecting the roller axis of rotation at its central point.

Omni wheel for this case is shown in Figure 1. There one can see the lateral view, fragment (a), of the wheel being equipped by four axisymmetrical rollers, each having a shape of the circular spindle. These rollers have been enumerated by their numbers. Each roller is connected to the wheel by a joint which axis coincides with the roller axis of rotation. These latter axes both are orthogonal to the wheel radius exiting from the central point O and passing through the the roller central point. So it is possible for the wheel to have a free rolling in direction perpendicular to its plane. Corresponding contacting curve with respect to the wheel coordinate system, being a circle in the case shown, has a coloured highlighting. This curve has a circular shape provided the wheel plane keeps its vertical orientation. Front view of the omni wheel is shown in fragment (b).

For the case of $\psi = 0$ being shown in Figure 1 a roller outer profile, generatrix, along its axis of rotation has evidently a circular shape, see Figure 1, fragment (a), again. This shape provides smooth transfer from one roller to another while the motion occurs. Evidently if $\psi \neq 0$ then it is not the case. Thus, the contact tracking algorithm for the case of $\psi = 0$ implemented in (Kosenko and Gerasimov, 2014) turned out to be simple enough. In the case of $\psi > 0$ it becomes visibly complicated. And its implementation on Modelica language is the main goal of this paper.

Other details of Figure 1 are the following: R is the omni wheel radius, R_1 is the distance between the wheel central point O and the roller central point, α is the half roller angular length from the viewpoint O . Unit vectors $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ of the base being connected with the wheel are shown in their initial positions.

In engineering applications one may encounter frequently a situation with $\psi > 0$. We proposed in (Kosenko and Gerasimov, 2014) fast algorithm for tracking a contact provided the omni wheel keeps vertical orientation of its plane (in frame of the whole vehicle construct). Thus the task for building up the contact tracking algorithm also for the case of $\psi > 0$ is of interest. This task

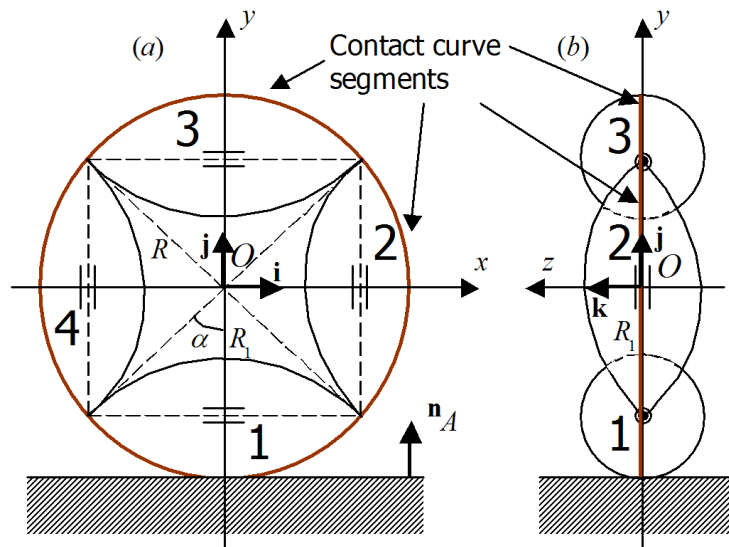


Figure 1. The omni wheel vertically aligned: (a) lateral view; (b) front view.

has been completed in this paper. To reach this goal we accept the working model of a virtual testbench consisting of one wheel equipped by rollers along its rim. One can see easily that this simplification has mostly methodical nature and does not prevent us from integrating all the construct back into the whole vehicle having generally several omni wheels previously analyzed (Kosenko and Gerasimov, 2014).

So let us consider an omni wheel, see Figure 1 its lateral and front views with four rollers, which is able to keep vertical orientation of its plane. We will see later how to arrange an implementation of such a servo-constraint. Note in addition, that in the case of $\psi > 0$ a generatrix of the roller outer surface will not be a segment of the circle anymore. It is represented by a more complicated curve. Moreover, point break of contact on the roller surface does not correspond to the surface tip for the case of $\psi > 0$ as it took place for the simple case of $\psi = 0$. To arrange correct simulation on event of the contact exchange between rollers one has to truncate the roller surface properly.

2 Model of the Omni Wheel Dynamics

Vehicle equipped by omni wheels might be replaced by a wrench consisting of force and torque in the multibody, rigid, representation. The force supposed to act at the wheel center. Thus approximately we can analyze the omni wheel dynamics with the wrench applied instead of a remainder of the vehicle.

Moreover, the vehicle, or a separated wheel, performs in our example motion on the horizontal floor for simplicity. Thus, the wheel being embedded into the vehicle in the simplest case should be aligned vertically. To ex-

press such an alignment analytically we can connect with the wheel the base $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ originating from the wheel center. Both unit vectors \mathbf{i}, \mathbf{j} lie in the wheel plane, and unit vector \mathbf{k} is normal to it. Thus the vertical alignment of the wheel is equivalent to horizontal alignment for the vector \mathbf{k} . Analytical condition for this is

$$\mathbf{k} \cdot \mathbf{n}_A = 0,$$

where unit vector \mathbf{n}_A is vertical, or normal to the floor. In other words, let $T \in SO(3)$ be the matrix of transformation from base $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ to the inertial absolute coordinate system. Then components of vector \mathbf{k} are exactly the components of the matrix $T = (t_{ij})_{i,j=1}^{i,j=3}$ third column. Thus one can express condition of the wheel vertical alignment in the form

$$t_{23} = 0.$$

This latter equation shows that the omni wheel multibody system undergoes the geometrical servo constraint. It is easy to see that this constraint may be implemented via control effort, rotating torque \mathbf{M} directed such as to prevent rotation of the wheel plane w. r. t. horizontal line belonging to this plane.

For details of the torque vector \mathbf{M} computation note that this vector has to be directed along horizontal line passing through the wheel center and belonging to its plane. Directing unit vector \mathbf{l} for this line has to satisfy the equation

$$\mathbf{l} = \mathbf{k} \times \mathbf{n}_A / |\mathbf{k} \times \mathbf{n}_A|.$$

Hence

$$\mathbf{M} = \lambda \mathbf{l}$$

and the multiplier λ is simply a value of torque balancing the wheel vertical orientation. In the wheel model torque \mathbf{M} has to be added to other torques applied to the wheel

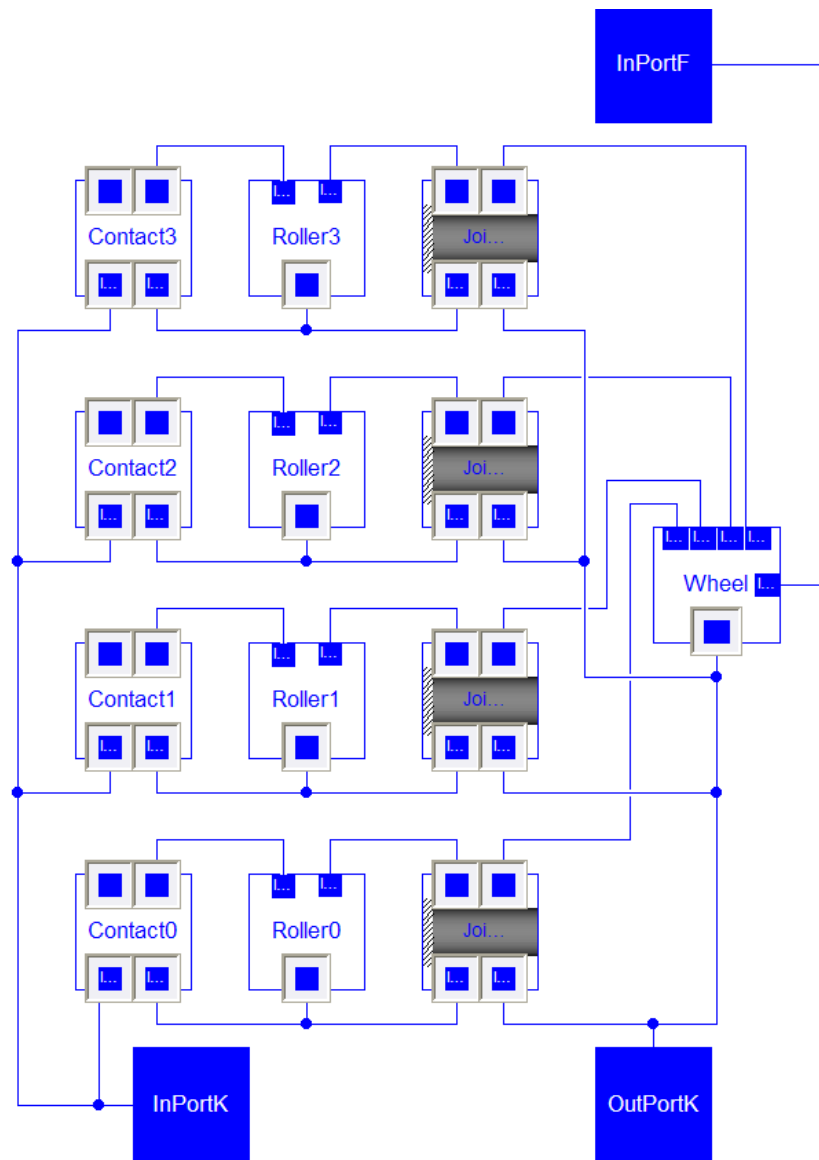


Figure 2. The omni wheel dynamics visual model.

under simulation. The value λ is exactly the Lagrange multiplier corresponding to the servo-constraint above.

It is easy to see the servo-constraint plays here a role of the virtual testbench for investigating the omni wheel dynamics. The remainder of the whole vehicle model is replaced simplistically by the wrench being applied to the wheel. The whole omni wheel dynamics visual model is seen Figure 2

As one can detect here the model of the omni wheel multibody system has been implemented using original multibody dynamics class library developed previously (Kosenko, 2005; Kosenko et al., 2006). One can use this library independently or with help of the known Modelica Standard Multibody class library or with any other Modelica library. The better way being recommended for such use is the following one. Firstly, one can implement mechanical subsystems of the whole sys-

tem under implementation. For instance, mechanisms having tree structure are modeled in a better way using Modelica Standard Multibody Library while mechanical subsystems including unilateral constraints with friction are better implemented using the aforementioned library of classes. Secondly, the only issue remained is to implement proper interfaces using models of ports mapping corresponding signals being transferred from one subsystems to another.

3 Implicit Contact Tracking Algorithm

We will assume in the further course that the wheel plane keeps its vertical orientation permanently. We have to introduce auxiliary orthonormal bases: $b_1 = \{\mathbf{i}_1, \mathbf{j}_1, \mathbf{k}_1\}$

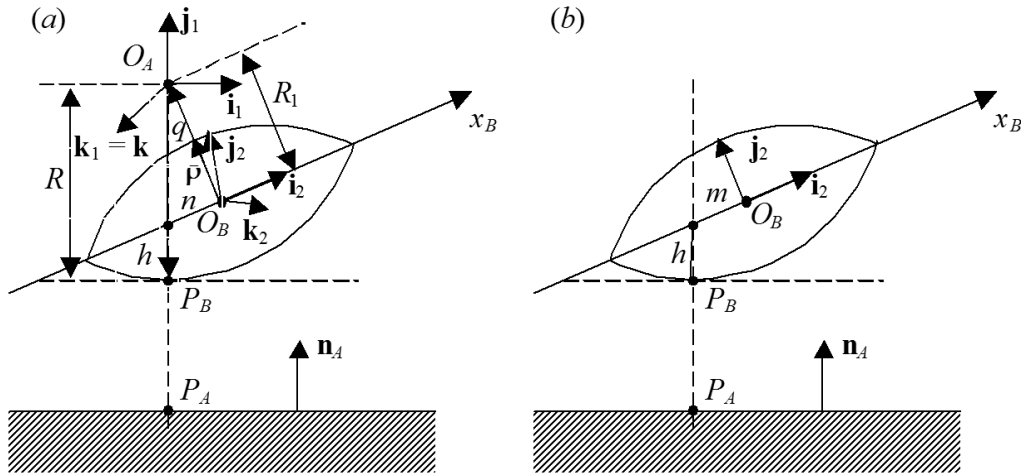


Figure 3. Contact tracking scheme: (a) lateral view of the omni wheel with a roller has been rotated about line $O_A O_B$ by the angle ψ , (b) lateral view of the individual roller.

and $b_2 = \{\mathbf{i}_2, \mathbf{j}_2, \mathbf{k}_2\}$. Intermediate base b_2 characterises partially position and orientation of the roller, while the base b_1 relates to the omni wheel.

The base b_2 coordinate system has its origin O_B at the roller central point. The unit vector \mathbf{i}_2 is directed along the roller axis of rotation, see Figure 3, fragment (a). The unit vector \mathbf{j}_2 is directed orthogonally to \mathbf{i}_2 and lies simultaneously in the vertical plane. The third unit vector \mathbf{k}_2 of the base b_2 is defined in a natural way as

$$\mathbf{k}_2 = \mathbf{i}_2 \times \mathbf{j}_2.$$

Remind here that all unit vectors are computed w. r. t. given fixed (absolute) coordinate system. We assume that positions and orientations are known for all bodies belonging to the multibody system for any instant $t \in [t_0, t_1]$ of simulation process. Therefore, we have

$$\mathbf{i}_2 = T_B \cdot (1, 0, 0)^T, \quad \boldsymbol{\rho} = (\mathbf{r}_{O_A} - \mathbf{r}_{O_B}) / |\mathbf{r}_{O_A} - \mathbf{r}_{O_B}|,$$

where T_B is the roller current orientation matrix.

Origin of the base b_1 coordinate system is located at the point O_A ($= O$ in Figure 1) of the wheel center. The unit vector \mathbf{i}_1 is oriented horizontally and belongs to the wheel plane. The unit vector \mathbf{k}_1 is orthogonal to the wheel plane and is identical to one of the wheel connected base vectors. We assume that using a controller the vector $\mathbf{k}_1(t)$ known we also have $\mathbf{j}_1(t) = (0, 1, 0)^T$ and $\mathbf{i}_1(t) = \mathbf{j}_1(t) \times \mathbf{k}_1(t)$.

Consider now relations providing base b_2 construction. Unit vector \mathbf{i}_2 has been built above. During roller and the floor contact the vector $\mathbf{i}_2(t)$ can not become vertical. Moreover, if the roller distortion takes place, its angle of rotation $\psi > 0$ about axis $O_A O_B$ is fixed non-zero, then the condition $\mathbf{i}_2 \neq (0, 1, 0)^T$ is fulfilled permanently. So we can assume that the condition

$$\mathbf{c} = \mathbf{i}_2 \times (0, 1, 0)^T \neq \mathbf{0}$$

is also fulfilled.

Thus, we can define $\mathbf{k}_2 = \mathbf{c}/|\mathbf{c}|$. And after this we can set $\mathbf{j}_2 = \mathbf{k}_2 \times \mathbf{i}_2$. Geometrical constraints, conditions of orthogonality to be exact, play important role in the omni wheel kinematics

$$\boldsymbol{\rho} \cdot \mathbf{i}_2 = 0, \quad \boldsymbol{\rho} \cdot \mathbf{k}_1 = 0.$$

These equations actually apply to computing the unit vector $\boldsymbol{\rho}$ and we have their differential versions

$$\frac{d}{dt} \boldsymbol{\rho} \cdot \mathbf{i}_2 + \boldsymbol{\rho} \cdot \frac{d}{dt} \mathbf{i}_2 = 0, \quad \frac{d}{dt} \boldsymbol{\rho} \cdot \mathbf{k}_1 + \boldsymbol{\rho} \cdot \frac{d}{dt} \mathbf{k}_1 = 0.$$

The value $c_\beta = \cos \beta = \mathbf{i}_2 \cdot (0, 1, 0)^T$ of cosine for the angle β of the roller axis inclination to vertical $(0, 1, 0)^T$ plays also an important role in the contact tracking algorithm. If current value of the variable c_β is less than some limiting parameter $c_{\beta \max}$, and simultaneously if an altitude of the point O_B defining position of the roller center is less than value R of the wheel radius then the contact takes place. Otherwise no contact occurs.

Note here that in order to arrange the unilateral constraint in the multibody system dynamics model the developer usually has to implement anything like hybrid automata construct. In our omni wheel model, on the contrary, this is not the case. It turned out sufficient to implement “simple” “if” construct to switch states “contact” and “no contact” for each individual roller, and simultaneously to advance forward “contact” state from one roller to its neighbour. The whole picture looks like from time to time neighbouring rollers mutually exchange their states. One can find details of the unilateral constraint implementation in (Kosenko and Gerasimov, 2014). Merely note that “if”-alternatives are the following: (a) “contact” state corresponds to zero-valued relative acceleration of two contacting surfaces at the point of contact, (b) “no contact” branch corresponds to

the zero-valued reaction mutual for both bodies at contact. All this is according to the Signorini rule. “if”-condition depends on the roller orientation variables.

Essential role in all these computations plays a contact tracking algorithm. Generally, its implementation reduces to computation of the contact point/patch which enables computing forces at contact. Usually, one considers contact of two surfaces participating in rigid/elastic interaction of two massive bodies. As a rule, such algorithms are pretty expensive and noticeably slow the whole simulation process. Fortunately, in case of omni wheels we found here the simplest way to make this computation as fast as possible using “elementary” geometric considerations.

We can also easily see from the Figure 3 that the point P_B of contact between roller and floor is obtained using formula

$$\mathbf{r}_{P_B} = \mathbf{r}_{O_B} + R_1 \boldsymbol{\rho} - R_1 \mathbf{j}_1 + \mu \mathbf{k}_1,$$

where the scalar μ is to be computed. Here the value R_1 is the distance between points O_A and O_B . The scalar μ can be computed if we multiply the last equation by \mathbf{k}_2 using dot-product. Thus we have

$$\mu = [R_1 \mathbf{j}_1 \cdot \mathbf{k}_2 - R_1 \boldsymbol{\rho} \cdot \mathbf{k}_2] / \mathbf{k}_1 \cdot \mathbf{k}_2$$

since the vector $\mathbf{r}_{P_B} - \mathbf{r}_{O_B}$ lies in vertical section of axisymmetrical surface of the roller, and the vector \mathbf{k}_2 by construction is orthogonal to this section. As a result the position \mathbf{r}_{P_B} of the contact point P_B is uniquely computed.

4 Explicit Contact Tracking Algorithm

Yet another way to obtain current position \mathbf{r}_{P_B} of the contact point P_B , or more accurately: the roller point closest to the floor, is an application of the following chain of equations. This chain is simply understood from geometrical scheme shown in Figure 3, (a) and (b).

$$\mathbf{r}_{P_B} = \mathbf{r}_{O_B} - m \mathbf{i}_2 - h \mathbf{j}_1,$$

where $m = R_1 \sin q / \cos q / \cos \psi$, $h = R - R_1 / \cos q$, q is the current value for angle of deviation of the vector $\boldsymbol{\rho}$ from direction of the vector \mathbf{j}_1 . So we have

$$\cos q = \boldsymbol{\rho} \cdot \mathbf{n}_A, \quad \sin q = (\mathbf{n}_A \times \boldsymbol{\rho}) \cdot \mathbf{k}_1.$$

Here we give explanations of some details of Figure 3. Fragment (a) corresponds to the lateral projection of the wheel and likewise the distorted roller projection. This latter object is shown here in a general position. Furthermore, P_B is the current contact point between the roller and the horizontal floor, n is a projection of the roller axial line segment onto the wheel plane. We can see easily that this projection is computed by the formula $n = m \cos \psi$ because the roller axis is turned about $O_A O_B$

by the angle ψ , see fragment (b) for the roller axial vertical lateral section. Thus, we have to pass two straight line segments from the roller center O_B to reach the point P_B : (a) the segment of the roller axis of length m ; (b) the segment down the vertical of length h . As we already mentioned above all variables needed are computed through known variables using explicit formulae.

In case of $\psi > 0$, distortion exists, for both implicit and explicit algorithms not all the length of the roller surface generatrix is necessarily in contact. So really we have to cut tips of rollers to provide regular simulation process. Length of the tip to be cut we can obtain for instance empirically or compute it explicitly. Indeed, one can easily see from Figure 3 that the real roller length should be computed by the formula

$$L = 2R \sin \alpha / \cos \psi.$$

“Ideal” switching of contact takes place in this case: exactly at the instant of contact loss for current roller a contact immediately arises for the “next” roller in direction of the wheel rolling.

5 The Wheel Model Classes Hierarchy

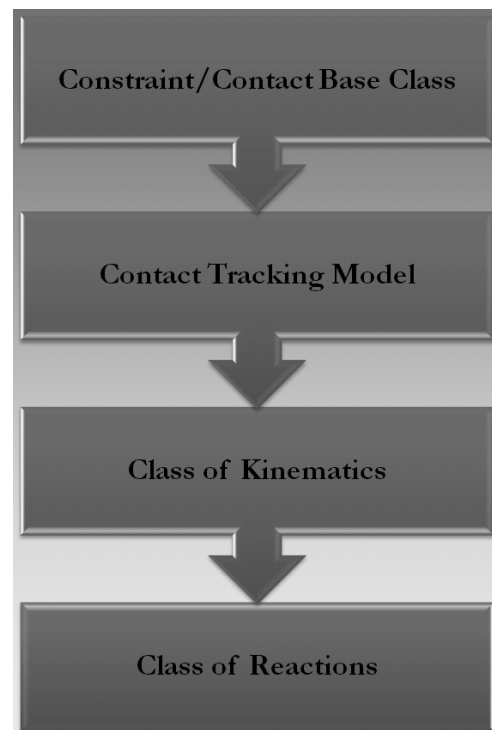


Figure 4. Contact model by stages of inheritance.

Model of the omni wheel testbench virtual prototype is a container class including the following objects instantiated: (a) disk of the wheel; (b) objects of rollers

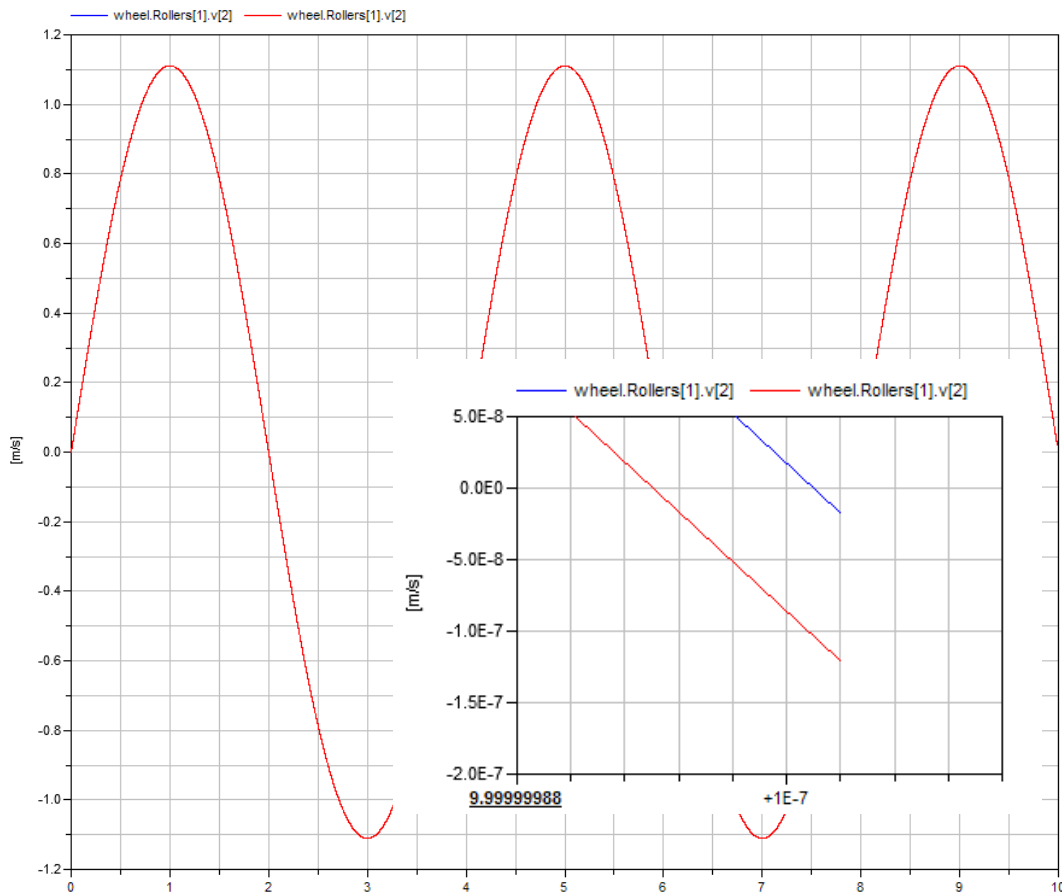


Figure 5. Comparison of dynamics for the roller No. 1 central point, its velocity y-coordinate, for cases of: explicit (blue curve) and implicit (red curve) algorithm of contact tracking.

mounted along the wheel rim; (c) objects of joints connecting rollers and the wheel disk; (d) objects of contacts connecting objects of rollers and the object of the horizontal floor surface; (e) model of base body as a horizontal floor.

Let us analyse in more details a structure of contact model. This model has many similarities with contact models previously considered (Kosenko, 2005). Nevertheless important differences exist. One of them mentioned above with regard to organization of the contact class using simple and efficient construct (Kosenko and Gerasimov, 2014). Note that in case of $\psi > 0$ the point of contact creates a curve with discontinuities at instances of rollers changes. However, this circumstance does not prevent the process of regular simulation.

Finally, we apply rigid point contact model as part of the simplest omni wheel model. For this we use the base class for constraint/contact models having only equations of Newton’s third law as a behavioral section (Kosenko et al., 2006).

We use class of the contact tracking model on the second stage of inheritance, see Figure 4. Cases of this class organization have been analysed above. Coordinates of nearest points P_A and P_B at contact for each pair (floor, roller) are computed as a result for this class functional-

ity.

Class for computing all kinematical characteristics at contact needed “works” in case of contact existence on the next stage of inheritance. On the third stage class for computing the reactions at contact is “turned on”. Reactions are the following: (a) normal reaction; (b) tangent force of friction; (c) torque of reactions (zero in the current consideration though it is not difficult to compute torque for several contact models).

To verify an approach for building up the models under analysis we compare the omni wheel dynamics in cases of implicit and explicit algorithms. The wheel performs free motion (combining rotation and sliding) with the only restriction: keep vertical alignment of the wheel disk.

Roller No. 1 central point, its mass center, altitude was analyzed and verified. More accurately we examine y-coordinate of the point velocity. Both models turned out almost identical: in the worst case we have a divergence: in accelerations of order 10^{-8} , in velocities of order 10^{-7} , in position of order 10^{-6} , over the time span being equal to 10 units of time. Results of simulation for velocities are shown in Figure 5. Other divergencies for the roller No. 1 central point acceleration and position at time = 10units are shown in Figure 6 and 7 respectively.

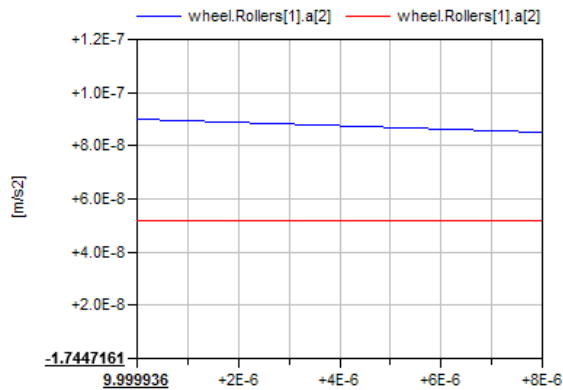


Figure 6. Divergence for y-components of acceleration.

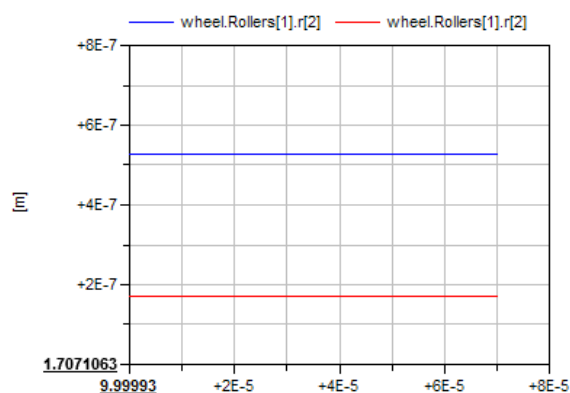


Figure 7. Divergence for y-components of position.

As expected the model with explicit contact tracking algorithm is faster approximately in 1.5 times.

6 Conclusions

The following effects were found as a results of new contact tracking algorithms applying to the omni wheel multibody system:

1. Two contact tracking algorithms were proposed: implicit and explicit. As expected the second algorithm turned out to be faster almost in 1.5 times. Both algorithms are simple (and efficient) even in simpler case of rollers without any distortion.
2. In case of distorted rollers contact curve becomes discontinuous at instants of rollers change. But simulation process maintains its regularity.
3. Both algorithms generate identical dynamics.
4. Process of the contact model design using technology of “vertical separation” outlined above has an

evident motivation and allows a simple generalization both for the normal force computation and for the tangent friction force model.

7 Acknowledgement

The investigation was performed under financial support provided by RSF, project 14-21-00068.

References

- B. E. Ilon. Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base. Technical report, US Patents and Trademarks office, Patent 3,876,255, 1975.
- V. Kálmán. Controlled braking for omnidirectional wheels. *International Journal of Control Science and Engineering*, 3(2):48–57, 2013.
- I. Kosenko and K. Gerasimov. Implementation of the omni vehicle dynamics on Modelica. In *Proceedings of the 10th International Modelica Conference*, pages 311–322, March 2014.
- I. I. Kosenko. Implementation of unilateral multibody dynamics on Modelica. In *Proceedings of the 4th International Modelica Conference*, pages 13–23, March 2005.
- I. I. Kosenko, M. S. Loginova, Ya. P. Obratsov, and M. S. Stavrovskaya. Multibody systems dynamics: Modelica implementation and bond graph representation. In *Proceedings of the 5th International Modelica Conference*, pages 213–223, September 2006.
- J. Tobolár, F. Herrmann, and T. Bünte. Object-oriented modelling and control of vehicles with omni-directional wheels. In *Computational Mechanics 2009*, November 2009.

Kansei Modeling for Delight Design based on 1DCAE Concept

Koichi Ohtomi

The University of Tokyo, JAPAN, koichi.ohtomi@delight.t.u-tokyo.ac.jp

Abstract

The main task of product development is to develop a good product at lower cost and to bring it to market in a shorter period. Conventional computer-aided design and computer-aided engineering (CAD/CAE) systems are well established in this regard. However, although upstream design is particularly important in product development to add value and incorporate the required functions, it is difficult to apply conventional shape-based CAD/CAE systems to the upstream design stage due to the lack of design information at that stage. As a solution to this issue, we are proposing the development of a design framework called “1DCAE”, which can be applied to the early design stage of product development including the conceptual and functional design phases. The 1DCAE concept can be applied not only better design, must design, but delight design. Here we introduce how 1DCAE concept applies to delight design and its core technology of *kansei* modeling.

Keywords: 1DCAE, delight design, kansei, modeling, better design, must design, CAD/CAE

1 Introduction

The 1DCAE based on a simple but an essential model is a methodology, a method and a tool to support the whole design from the early design stage to the detail design stage and to glance from mechanical, electrical to software design. 1D means to capture the essence of things including physical phenomena and to express by simple model to be easy to understand. 1DCAE can realize the evaluation by CAE from the upstream to downstream design. There are three kinds of design fields to apply 1DCAE concept. They are better design to realize the low cost and better performance, must design to realize the safety and security, and delight design to get to customers’ heart. Delight design has become more important than ever in manufacturing with the diversification of recent customer requirement. On the other hand, we need to model the characteristics of *kansei* in delight design in addition to the conventional physical model for better and must design. *Kansei* is originally a Japanese word that refers to the sensitivity of a human sensory organ at which sensations or perceptions take place in response to stimuli (e.g., a product) from the external world. Kansei includes evoked senses, feelings, emotions, and impressions. The word *kansei* has begun to be used internationally

because there is no suitable translation in English. Figure 1 shows the image of *kansei*.

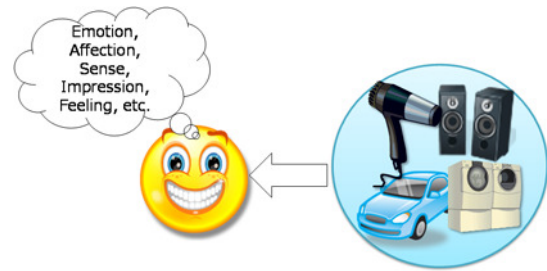


Figure 1: what is *kansei*?

Here we call the model to express the characteristics of *kansei* “the *kansei* model”. *Kansei* modeling based on 1DCAE can introduce *kansei* into the flow of manufacturing and provide a platform of delight design that does not depend on intuition and experience. *Kansei* model is implemented on the 1D tool based on Modelica language. *Kansei* model enables quantitative evaluation because it starts from the physical model.

2 Concept of 1DCAE

Product development starts in a conceptual design and progresses with functional design, a layout design, a structural design, and a manufacturing design. However, in the early design stage of design process design information is ambiguous. Therefore, it is difficult in the early design stage of a design to apply conventional design tool such as CAD/CAE. In a second half of design process CAD/CAE will become applicable, but in this stage, there are many design restrictions and then flexibility of design decreases. Moreover, not only CAD/CAE but rapid prototyping (RP) and experiment are possible in the second half of design process. Therefore, the design methodology/method to be applicable from the early stage of design process is desired.

In the matured industrial field, it is sufficient for a design to start from a layout design and a structural design, but in the industrial field that is strongly depends on the voice of customer (VoC), it is required to start from the upper stage of design process to get the VoC before starting a detailed design.

Then, we propose the 1DCAE concept to be applicable from the early stage of design process. “1D” doesn’t

mean one dimensional, but it means simple but essential expression of phenomena. By applying 1DCAE, it becomes to be able to evaluate from the upper design stage to the lower design stage by CAE. The CAE mentioned here means not a simulation but the original concept of Computer-Aided Engineering.

In case of a product design by 1DCAE, the concept of product and/or system should be expressed in functional basis and then can be analyzed by system simulation tool based on Modelica before creating a shape. This process enables the totally appropriate design in the early stage of design process. The output of 1DCAE is the input of CAD/CAE. Figure 2 shows the design process based on the 1DCAE concept.

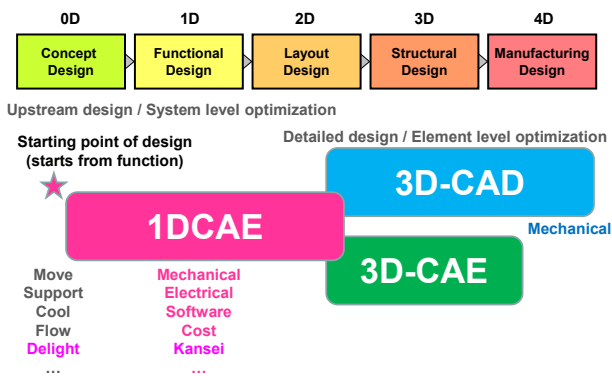


Figure 2: Design process based on 1DCAE concept

The relation between 1DCAE and 3D-CAE is shown in Figure 3. In 1DCAE, the target of a product development is set up and a conceptual design and a functional design are performed. By considering the function of a product, the preliminary decision of the design specification is carried out, and it delivers to 3D-CAE. In 3D-CAE, a structural design and a layout design are performed based on the specification received from 1DCAE. 3D-CAE is a part to consider structure, and a conventional CAD/CAE demonstrates their power. The result of 3D-CAE is fed back to 1DCAE and performs functional verification as a system. The 1DCAE in a broad sense is a design framework combination of 1DCAE and 3D-CAE.

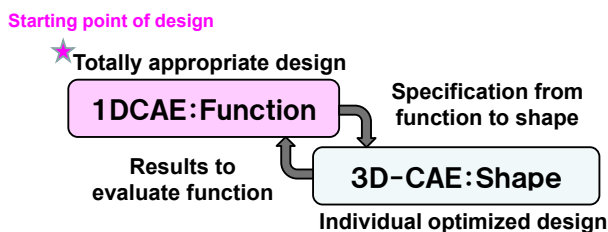


Figure 3: Relation between 1DCAE and 3D-CAE

3 1DCAE and delight design

The 1DCAE has a different configuration depending on the purpose of design. We classify into three kinds of design depending on the purpose. Three kinds of design are classified according to Kano model as shown in Figure 4.

- I. Must design equivalent to commonplace quality: Design for assurance design. Many of the trouble caused by neglecting this design. It is not so easy to evaluate the designers' effort for this design, but the basics of design.
- II. Better design equivalent to performance: Because the target is so clear, it is easy to approach for this design. On the other hand, it will fall into cost competition eventually. The purpose of this design is to cost minimization, time to market (TTM) minimization, and performance maximization.
- III. Delight design equivalent to attractive quality: Design concept is the most important in this design. Many of hit products come from this area.

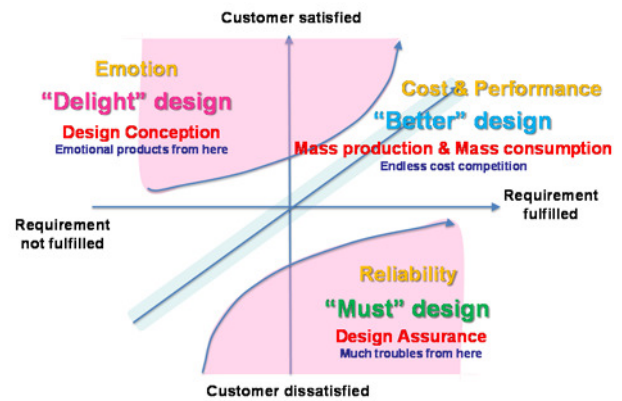


Figure 4: Three kinds of design

It is crucial to produce emotional products (delight design) in addition to must design and better design in the future industry. Therefore we focus on the delight design in this paper. Figure 5 shows an example of 1DCAE delight design to apply home appliance. Conventionally, product sound design focused on reducing the noise level. Here we propose another aspect of product sound design to add values to products. To capture the sound as a value rather than noise, it requires the extraction of the potential needs of customers on the sound. Because customer needs are diverse, it is important to set the target in consideration of this effect. On the other hand, there exists a design metrics of noise level in case of noise reduction. When we treat the sound as a value, we need to introduce new metrics to express comfortable sound as designs can understand. 1DCAE delight design to product sound design is to the process to extract customer needs, develop the metrics of sound, set the target sound, and realize as products. Product design performed based on the requirement derived from 1DCAE process, sound

data from prototyping mapped on the metrics of sound to validate the performance, and finally we got the new product to satisfy the target sound with comfortableness. Because delight design was conducted from the early stages of product development together with must design and better design, there was no additional cost and no negative impact on performance.

Delight design treats kansei. Kansei is also one of the functions of products, but it is difficult to combine function and structure directly. Therefore, we combine psychological domain function (worth) with structure via physical domain function of sound quality metrics as shown in Figure 6. Sound quality metrics which was created as a result of the field of psychoacoustic, and can represent the perception about the human sensitivity to sound as objective numerical value that was analyzed as sound data. As shown in Figure 6, loudness, sharpness, roughness, and fluctuation strength are often used as a representative sound quality metrics. These metrics enable us to express comfortable sound as a common language. In addition, sound quality metrics will be kansei metrics for sound, which will be described later.

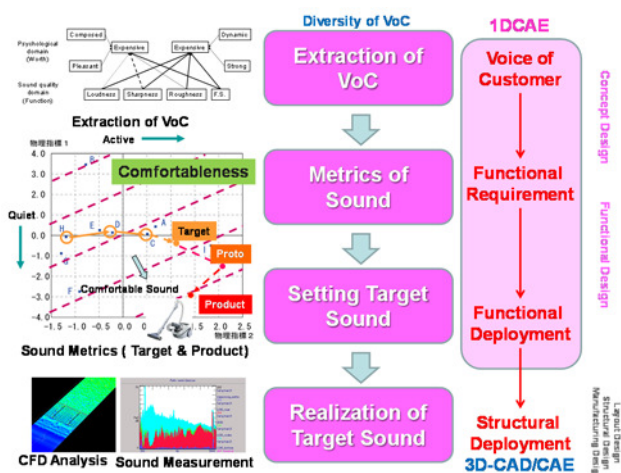


Figure 5: Example of delight design based on 1DCAE

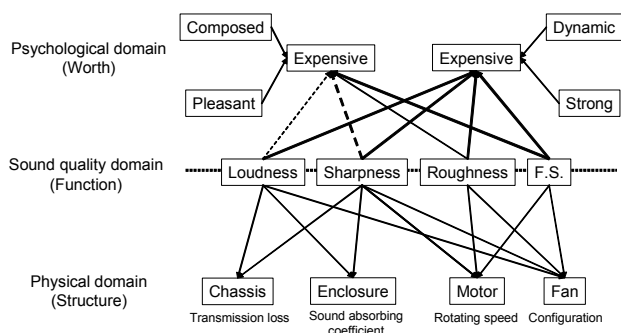


Figure 6: Combination of psychological domain with physical domain via sound quality domain

4 Kansei modeling based on 1DCAE

We introduce the kansei modeling based on 1DCAE concept to realize delight design. The technology to capture the kansei is defined as the kansei modeling and the resulting model as the kansei model. 1DCAE is done by using the so-called 1D tool based on Modelica language. Figure 7 shows a model image of the hair dryer in which the left half is for the physical model and the right half for the kansei model. Starting from the physical model, then kansei model is built by cooperation with the kansei database. In case of dryer, air flow, sound, and handling are related to kansei.

Designers perform the delight design by using 1D tool with kansei model. They can check the degree of attainment for their ideas by calculating the attractive metric on 1D tool. Results of delight design obtained in this process is sent to the mechanical design and circuit design processes to perform the tangible design.

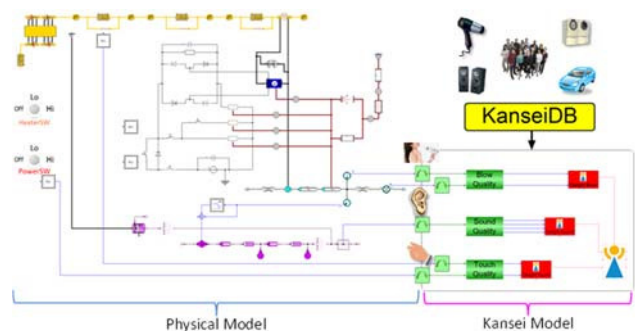


Figure 7: Image of kansei modeling

5 Kansei modeling in case of sound design

Then, we will introduce the kansei modeling in case of sound design. The physical model is derived by analyzing the existing product as truly as possible considering the physical law. Figure 8 shows the 1D physical model of a dryer. The basic elements such as a fan, a motor, and a heater and their related structure, mechanism, housing, electronics, and software are expressed by using the 1D tool software.

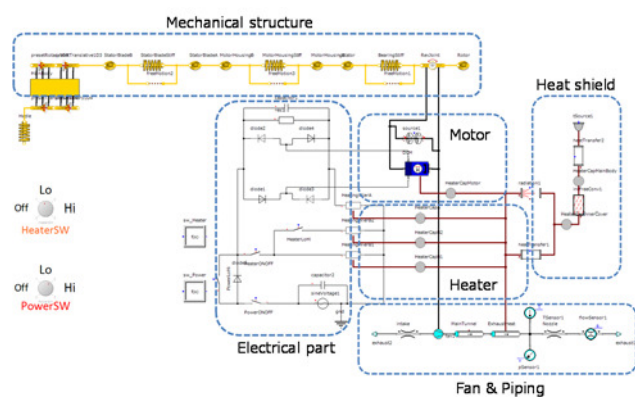


Figure 8: 1D physical model

Figure 9 shows the FS (function and Structure) map in order to set the development goals of a dryer. The FS map informs us the design items related to must, better, and delight design. From the extracted delight design items, we selected the structure to create comfortable sound.

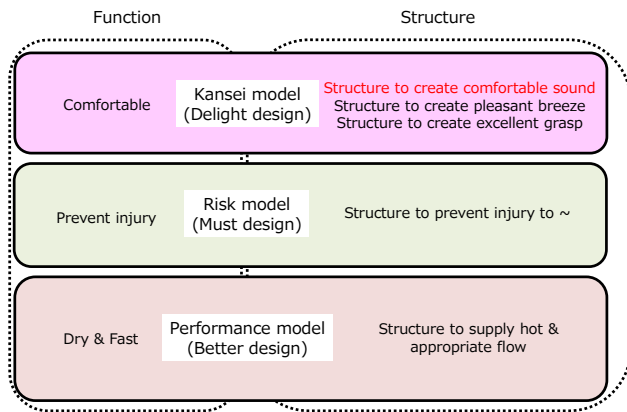


Figure 9: FS (Function Structure) map

The acoustic model should be added to the 1D physical model as shown in Fig.8 as the first step of the kansei model for sound. Figure 10 shows the 1D physical model with acoustics.

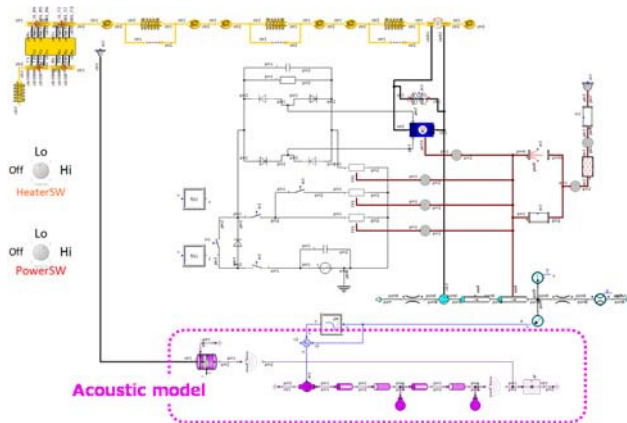


Figure 10: 1D physical model with acoustic model

Then we build the kansei model for sound as shown in Figure 11 according to the same procedure as shown in Figures 5 and 6.

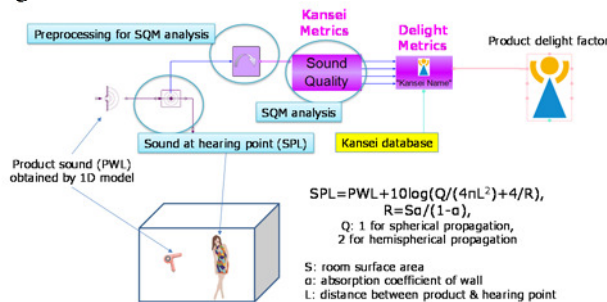


Figure 11: 1D kansei model for sound

Figure 12 shows the example of simulation results to describe the sound quality metrics and FFT when a dryer starts to operate from power off condition to low mode and then high mode.

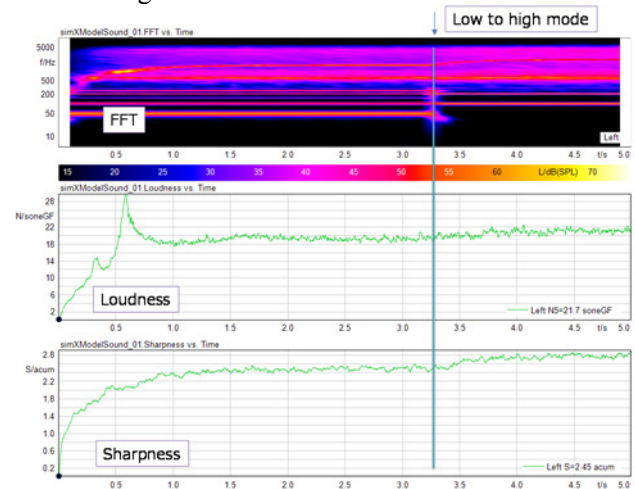


Figure 12: Example of simulation results

6 Future work

We aim at the delight design for kansei in general. In the future, kansei models not only for sound quality but for visual quality, flow quality, and touch quality will be developed as shown in Figure 13.

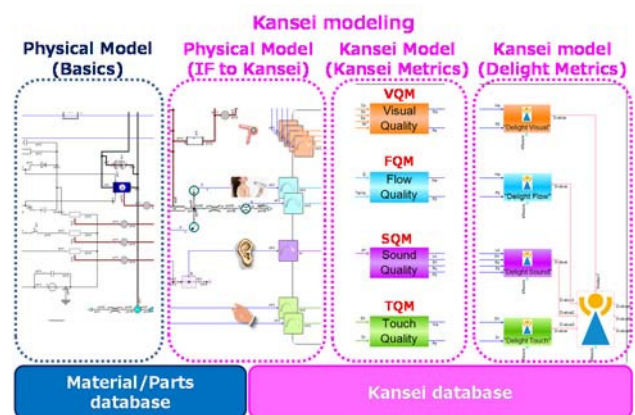


Figure 13: Total image of 1D kansei model

7 Conclusion

We introduced the kansei modeling based on 1DCAE concept. Kansei model was implemented on the 1D tool based on Modelica language in case of sound design. Kansei model enabled quantitative evaluation because it started from the physical model.

Acknowledgements

This research was supported by New Energy and Industrial Technology Development Organization (NEDO) of Japan, and we would like to thank them for their assistance.

References

- Ohtomi, K. and Hato, T., "Design Innovation Applying IDCAE", July, TOSHIBA REVIEW. (2012)
- Ohtomi, K., Design of Worth for Customer Product Development, What is "What's the Design"? Special Issue of Japanese Society for the Science of Design vol.16-2 no.62, 31-38, 2009
- Ohtomi, K., Hosaka, R., 2008, "Design for product sound quality", Internoise2008.
- Ohtomi, K. and Hosaka, R., "Product Sound Design", September, TOSHIBA REVIEW. (2007)
- Ohtomi, K., "Importance of Upstream Design in Product Development and Its Methodologies", January, TOSHIBA REVIEW. (2005)
- Yanagisawa, H., Murakami, T., Noguchi, S., Ohtomi, K., Hosaka, R., 2007, "Quantification method of diverse kansei quality for emotional design application of product sound design", ASME DETC2007-34627.
- Zwicker., E, 2006, "Psychoacoustics: Facts and Models", Springer, 3rd Edition, Springer-Verlag, New York Inc.

A Modelica Library Organization Method Supporting Online Modeling and Simulation

Xiong Tifan¹ Zhou Zhiming¹ Wan Li¹ Li Yongchao¹

¹CAD Center, Huazhong Univ. of Sci & Tech, China,
{xiongtf, wanli}@hust.edu.cn, {zhouzm, liyc}@comodel.net

Abstract

Today, the trend of achieving networked collaborative innovation and design of complex product based on Modelica is predictable in the industrial field. However, the existing file-based Modelica library organization method designed for single-machine environment does not satisfy the model management requirements for dynamic collaborative modeling and sharing under the network environment. Aiming at this problem, a new organization method of Modelica library based on database is proposed. The main principle of this method is that the organization objects are models rather than files. Through interacting with database storing metadata describing models, it is available to achieve model management based on the granularity of single model. Finally, a network-based multi-domain unified modeling and simulation platform is developed on the basis of the model management architecture.

Keywords: Modelica, organization method, online modeling and simulation

1 Introduction

In recent years, with the increasing complexity of the products in the field of engineering, it is difficult to construct all sub-models from different areas for one person or one team (Zha, 2006), and separate subsystem simulation in different fields cannot meet the requirement of the design innovation. The process of modeling and simulation for complex products is moving in the direction of integrated multi-domain modeling. Modelica, as a unified object-oriented modeling language, can solve this problem properly well. However, the existing Modelica softwares of single-machine environment do not support sharing and reuse of models, which seriously slows down the process of product innovation. So in order to meet the challenges of collaborative management and sharing of models in the internet-distributed environment, it is extremely necessary to build an online platform to provide services for different user roles, such as one person, one team or one enterprise, as shown in Figure 1. A variety of intelligent solutions have been proposed to explore the network-based system.

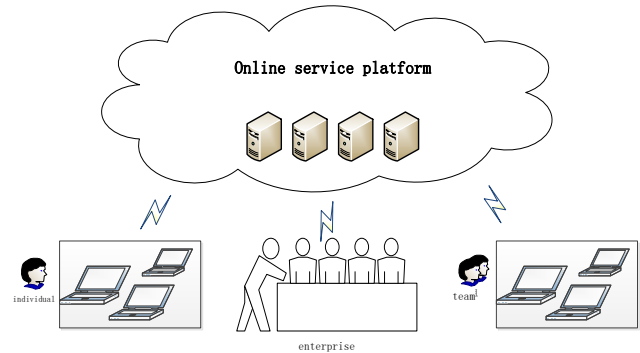


Figure 1. Online service platform

Eissen et al. discussed different realization alternatives for Web-based simulation services and presented the prototypic implementation of a web service which is built on the proposed W3C Web interface stack (Eissen, 2006). This research allows for the analysis and execution of technical models described in the well-known Modelica modeling language. Shi et al. presented an Internet-based electrical engineering virtual lab (IEEVL) using Modelica for unified modeling. It uses XML to represent and exchange information and is only capable of modeling and simulating for electrical engineering domain (Shi, 2011). Mohsen et al. constructed a web-based teaching environment, OMWeb, which is part of the open source platform Open-Modelica. It can be used both in engineering courses as well as for teaching programming languages (Mohsen, 2011). Oscar built the UN-VirtualLab, this web platform offers a free web simulation environment for educational purposes. Users can simulate models that have been stored on the server through setting parameters (Oscar, 2011). Zhang et al. researched and developed a web platform called Proteus. This platform is designed for education and academic research, and provides a place where students, educators and academic researchers can easily create and share their models of physical systems described using Modelica (Zhang, 2013).

However, though these research and platforms could help us to solve some problems in a certain extent, there exist serious obstacles. They do not break through the traditional static unstructured organization method of Modelica library, but just developed as

remote presentation systems using the mechanism of compiler and solving in the single-machine environment. The functions are too incomplete to get models reused and shared through the internet and do not support multi-domain collaborative online modeling.

This paper mainly aims to propose a decoupled structured dynamic model library organization method supporting control on single model by combining the database with file storage. The method adopts a way of one file corresponding to one model, to eliminate the coupling relationship among multiple models within a single file. So, through managing model metadata by database and organizing model files by file storage, it is available to restore the stand-alone environment on the server for the existing Modelica compiler and meet the functional requirement of re-use and sharing under the network environment.

2 Analysis of traditional Modelica library organization

In the Modelica language specification, the main compositions are consist of eight types, package, model, type, connector, block, function, record, and class (Fritzson,1998). In order to discuss conveniently, here we just define two types, *package* and *model*. The package of specification is our defined *package*, and the remaining types are collectively called as *model* type, because their operation and organization in the database and file storage are same in this paper.

As shown in Figure 2, all Modelica model libraries, both standard and private, are constructed in the form of *package.mo* files. The physical file organization of Modelica library is based on the file directory, including folders and *mo* files, and the logical file organization takes *Package* and *Model* as its objects. Modelica compiler is the core engine of a Modelica-based multi-domain physical modeling and simulation platform. When the compiler works, it needs to search and analyze the referenced or inherited models of current model, and referenced or inherited models in a deeper level (Zhao, 2011). The model library organization based on file directory can well support the existing compiler's searching and compiling function.

As to the traditional organization of model library, there are two methods, structured organization and unstructured organization (Modelica Language Specification Version 3.0). If the folder directory contain *package.mo* file, the organization is structured. In unstructured method, there is no *package.mo* file, package classes and sub-classes exist in the same mo file. For example, as shown in Figure 2, if the Rotational library is organized by the unstructured way, there is only a *Rotational.mo* file, and then it contains all the information, including sub-models, Spring and Damper, also including models in sub-library,

AngleSensor and SpeedSensor. A combination of two ways is applied to the existing software in the single-machine environment, which will lead to a situation that one mo file contains multiple packages or multiple models, as shown in Figure 3.

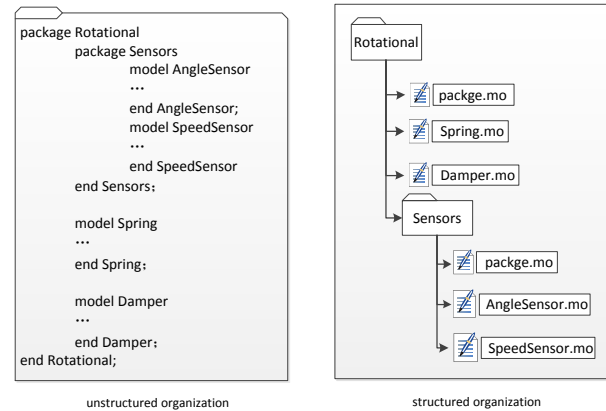


Figure 2. Traditional Organization of Model Library

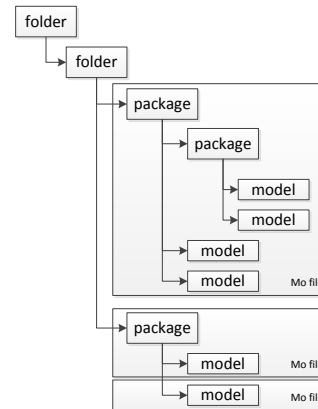


Figure 3. Model organization based on file index

According to the functional requirements of users, the online platform should supply the modeling service, simulation service and model management service at least. Modeling service provides visual modeling to help to create basic components and models. For completed models, users can run simulations by setting the parameters of components through the web-based simulation service. And the model management service mainly enables users manage their models conveniently, including uploading models from local environment, sharing models, renting models, and so on. So the current model management organization is too extensive to satisfy these demands. It mainly displays in two aspects:

1. Simple file directory mode. As to traditional Modelica model libraries, there are only mo files in the file storage, the mo files are taken as control objects. Though the mode meets the requirements well for the model-loading of the compiler, it is negative to achieve the control on single model in a unified way. To realize the sharing and control of single model and make it easy to query and

manage, we need to change the organization of the model library and organize it orderly based on its object feature attributes, such as the model's name, owner, creation time, release time, the release states, etc.

2. Unstructured model organization method. There can be coupling relationships in a single file, if the library is organized by this method. And you may change other models in a same file when you change one model, which is not suitable to manage models based on the granularity of single model.

3 Model Management Architecture

According to the analysis above, in order to achieve granularity management of single model in the network environment, it's required that the organization objects of library are models rather than files, and the feature attributes and parameters of single model and relationships between models, here collectively called as metadata of models, should be stored on the server. A relational database can help achieve this purpose. Under this premise, in order to obtain compiler's support to realize the online modeling and simulation, the single model in the matching relational database needs to get recognition to search corresponding mo file from a file system. So one to one mapping relationship between database and file base is required.

As shown in Figure 4, in the B / S architecture, we use a combination of model database and model file base to provide data support for modeling and simulation, users can interact with the browser to get the modeling and simulation service and model management service.

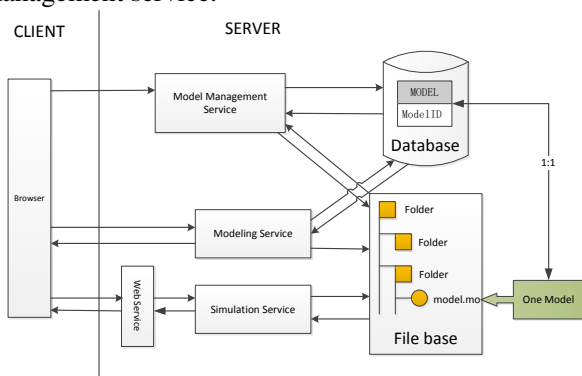


Figure 4. Model management framework

Database is used to store the basic properties (metadata) of model and the relationships (reference, inheritance, etc.) between the models. And file base is organized by *mo* files based on file directory to support the online compiling and simulating service. The model management service can be achieved on the basis of attributes of models in the database. For example, the display and renting authority of model could be accessed via simply changing publishing state. If this model has been published, it can be rented and referenced by other users to acquire re-use. Instead,

others cannot search it on the Internet. The modeling service saves the models' mo file in the file base and the models' metadata in the database. Besides, solving results can be stored in the file base by simulation service. In this framework, one model's descriptive information (metadata) in the database corresponds to one mo file in the file base. So, the management of model resources based on the granularity of single model can be reached.

4 Detailed design of management framework

The framework involves the database design, the file base design and the dealing with the relationship between the database and the file base. In addition, importing the existing libraries, managing library version and collaborative modeling are also worth researching. We will discuss these aspects in this part.

4.1 Database design

As shown in Figure 5, we design the metadata of model object which includes name (Name), creator (Creator), create date (CreateDate), release date (PublishDate), status (Status), text information (Text), model's parameters (Parameter), industry information (Industry), major information (Major), model price (Price), model description (Description) and so on. By the designing of these attributes, any model can be acquired, and any operation can be executed. The text information contains four parts: icon, diagram, information and used, which is used to displaying models in different views. Inheritance (Extend) and reference (Used) relations between models can be used to searching models. Package object's attributes are similar to model, which is included in package.

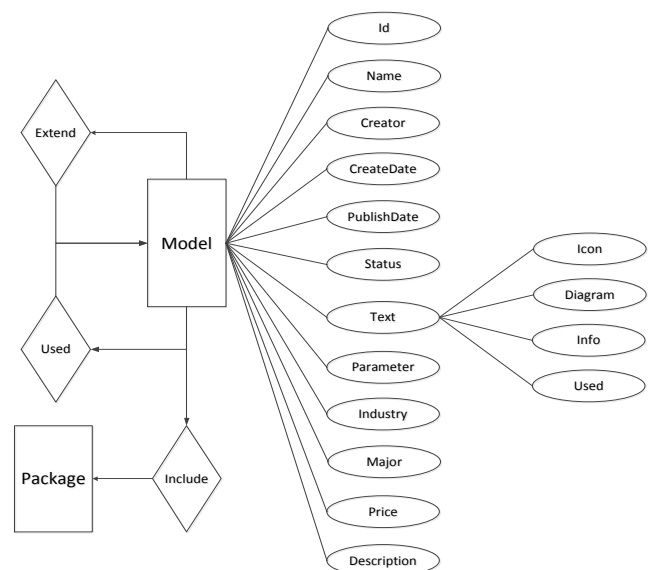


Figure 5. Design of model object

In order to achieve structured display in the client interface (an internet browser) and management toward

single model, it is necessary to use the database to manage the model library. In accordance with the model object, we separate sub-models' details and relationship from coarse-grained *mo* files into the database.

The database structure is shown in Figure 6, Model table, File table and SVG table are designed to save the basic information, text information and svg information (Scalable Vector Graphics, an XML-based language describing two-dimensional vector graphics in a graphical format for graphical modeling and interface icons) of the fine-grained models after initializing from the model library. And user attributes are added to achieve the user's management and control on models. The Component table stores components of all models. The Version table helps to achieve model version management. The Industry table and Major table can separate models into different industries and majors. The Parameter table saves parameter information o models. The Compile Task table records information of compile task and Simulation table records information of simulation task.

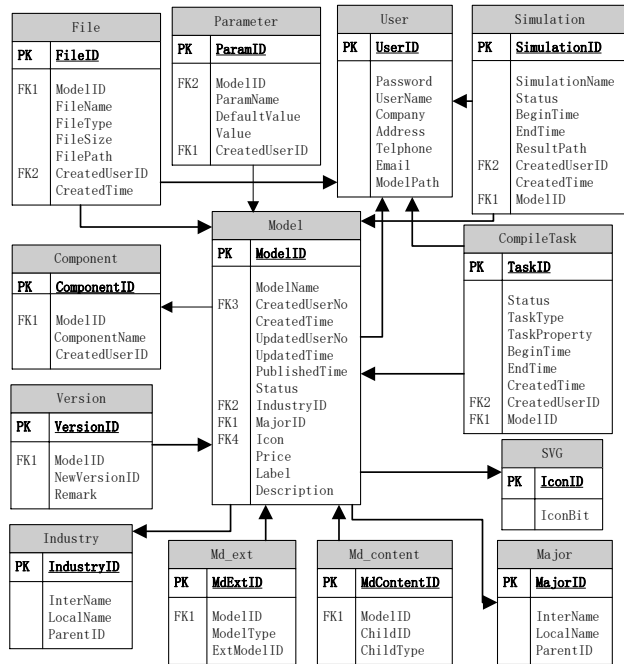


Figure 6. Database structure

Inheritance and reference information of models are critical. These information are included in *mo* file in the traditional organization method, and compiler can get these information by loading the file into the memory to have a complete model. There will be an error for server to compile a model's *mo* file if the information is incomplete, which will lead to an unfriendly user experience. For this reason, inheritance relational table (Md_Content) and reference relational table (Md_Ext) are added to store the inheritance and reference relationship between models. When a user loads a specific model, the server will load all the models associated with this model from the relational database.

Through the designing of database, one user can get all the corresponding information of himself or herself, including user information, models, compile tasks, simulation tasks, and so on. The authority control can be achieved conveniently.

4.2 File Base Design

The modified model library organization mode is shown in Figure 7. The physical organizational mode is still represented by folder and *mo* file, and also the logical organizational mode is still organized by package and model object. But the mapping relationships between models and packages have been changed. We abandon the former coupling relationship between models or packages in *mo* file, by using a single *mo* text to represent a single model or package.

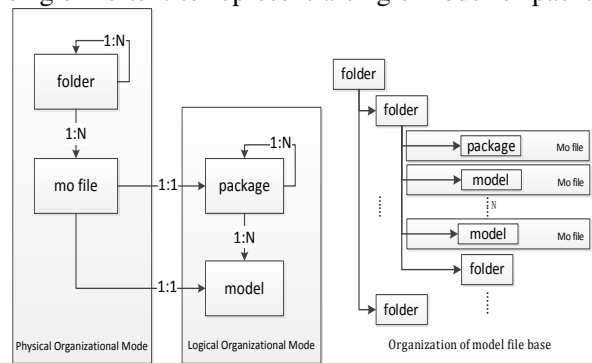


Figure 7. Mapping between models and packages

Compared to the existing model library organization mode, all the models separated from an original package will be stored in a folder named by the package's name, and the *package.mo* file will be the loading index of the compiler, which do not change the way of loading models based on file directory. So the existing compiler can be used in the internet-distributed environment.

In the database, model is the basic unit of management. And *mo* file is the basic organization object in the file base. After improving organization of Modelica library, one record of model in the database can be matched with one *mo* file in the file base, as shown in Figure 8. Then any operation to one model through database can respond to the *mo* file, like adding components, modifying parameters and creating equations. In this way, it is easy to realize synchronous management of database and file base in the process of online modeling.

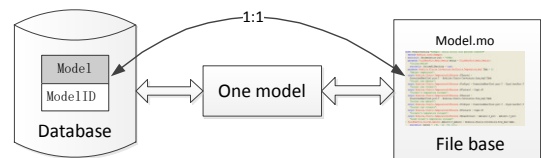


Figure 8. The relationship between database and file base

4.3 Dealing with existing libraries

There have accumulated so many model libraries, since the softwares in the single-machine environment, such

as Dymola (Dag, 2002), MWorks (Wu, 2006) and Openmodelica (Fritzson, 2005), having been used in machinery, electric, aerospace and other fields for more than a decade. The network platform should make full use of these libraries and import them to expand the library resources, which would be beneficial to achieving re-use, cutting costs and shortening the cycle of product development.

For the existing libraries, there are only mo files and unstructured coupling relationships are common. Then research on how to split them into single-type mode, and storing them in the database, would be crucial. The detailed process of dealing is shown in Figure 9. After uploading a library file, the splitter of the online platform analyzes every mo file, splits them into lower-level mo file one by one, until it just contains one package or one model, and creates metadata in the database about corresponding package or model at the same time. By this way, the existing Modelica libraries can be stored and reused well on the online platform.

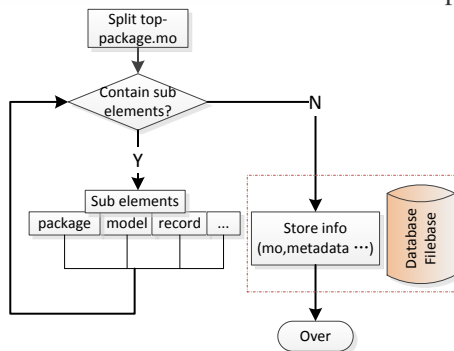


Figure 9. Process of splitting mo file

4.4 Collaborative modeling and version management

The version of one library is labeled as a number separated by ‘.’, like “3.0”. When a library is created, the version is defined as 1.0 by default. While one library is published, a copy of this library will be added into database and file base, whose state will be set as published, and meanwhile, the version of original library would be modified as 2.0, which are still editable for creator. This published library can be referenced by others, so the re-use of models is reached easily.

For one huge project, it is necessary to create a group or team on the online platform. All members of this group can edit this library, but different member would get different task according to their professions, and also they would have different authority of task. The complex model can be implemented through integrating sub-models of group members by a charge leader. The progress of collaboration is as shown in Figure 10, multi-domain collaborative modeling can be achieved through the division of task.

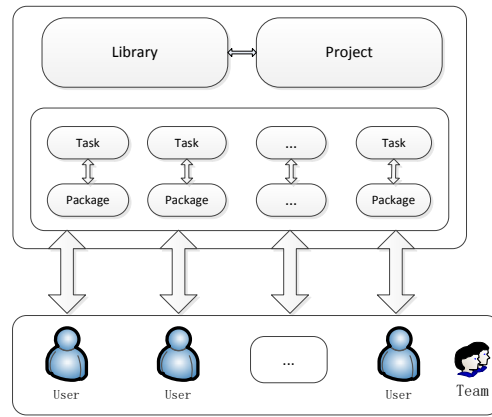


Figure 10. The progress of collaborative modeling

5 Application Verification

An online service platform supporting multi-domain physical modeling and simulation in the web environment - CoModel (<http://www.comodel.net>), as shown in Figure 11, has been researched and built based on this organization method of Modelica library.

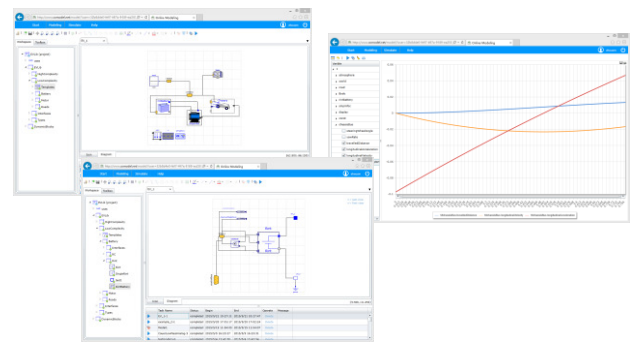


Figure 11. Online modeling page of CoModel

The architecture of data supporting and modeling service of CoModel platform is shown in Figure 12.

In the collaborative visualization modeling and simulation platform, users can interact with the browser to get the modeling and simulation service and model managing service. Modeling and simulation functions are on the basis of file directory in the file storage, which do not change the way of searching and loading models of existing compiler, can be implemented effectively. Solving results are also stored in file storage, users can download them easily. Model management can be achieved through contacting with database and file storage. The platform builds four kinds of libraries, including standard libraries, individual libraries, group libraries, and public libraries. Standard libraries are provided by Modelica Association (<http://www.modelica.org>). Individual libraries are created by a user. They store users’ private models and nobody can get the models’ information except the models’ owner. Group libraries are developed by a team, aiming at complex and collaborative product designing. Once individual libraries and group libraries are uploaded and

published on the server, they become public libraries, and then every user in the platform can rent them to establish their own models. All the libraries are organized by a combination of database and file storage. Database stores all the metadata about information of model and package, and file storage stores all mo files by file path. They would respond to

any operation on any model in the process of modeling. The fact that at present, this data supporting method based on the decoupled structured dynamic model library organization shows good performance on the platform, proves that this method is advanced and effective to support online modeling and simulation.

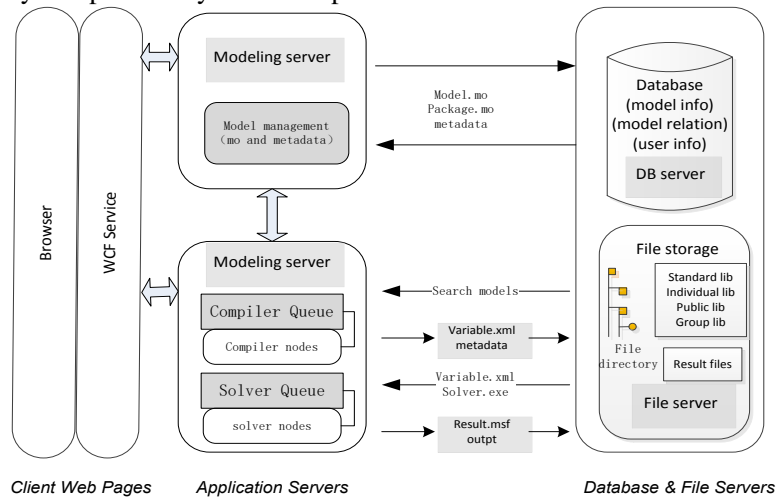


Figure 12. The architecture of CoModel

6 Conclusion

The library organization method acting as the data support of the web-based general multi-domain physical modeling and simulation can effectively achieve dynamic collaborative management and sharing of models under the network environment. And based on this method, the technical feasibility of the

reuse and redeveloping of model for further development can be increased. Of course, this work is just an initial study of the web platform supporting online modeling and simulation, and just a compromised way for existing compiler in stand-alone environment. The future work involves the development of networked compiler and improving the efficiency and performance of online platform.

References

- Modelica Association. Modelica – A Unified Object-Oriented Language for Physical Systems Modeling Language Specification Version 3.0. <http://www.modelica.org>.
- Brück Dag, et al. Dymola for multi-engineering modeling and simulation[C]. *Proceedings of Modelica*, 2002.
- Peter Fritzson, Vadim Engelson. Modelica – A Unified Object-Oriented Language for System Modeling and Simulation[J]. *Lecture Notes in Computer Science*, 1998:67-90. doi: 10.1007/bfb0054087.
- Peter Fritzson, Peter Aronsson, Lundvall Håkan, et al. The OpenModelica modeling, simulation, and development environment[J]. *Simulation News Europe*, 2005.
- Torabzadeh-Tari Mohsen, et al. OMWeb – Virtual Web-based Remote Laboratory for Modelica in Engineering Courses[J]. *Tari*, 2011. doi : 10.1109/ICCSN.2011.6013894.
- Duarte Oscar. UN-VirtualLab: A Web simulation environment of OpenModelica models for educational purposes[C]. *Proc 8th Modelica Conf*. Dresden , Germany: Link ping University Electronic Press, 2011: 30-31. doi: 10.3384/ecp11063773.
- Eissen S M Z, Stein B. Realization of web-based simulation services[J]. *Computers in Industry*, 2006, 57(3): 261-271. doi: 10.1109/VPPC.2006.364294.
- Zhengyin Shi, Shenglin Zhao, Shan-an Zhu. An Internet-based electrical engineering virtual lab: Using Modelica for unified modeling[J]. *IEEE International Conference on Communication Software & Networks*, 2011:555 - 559. doi: 10.1109/ICCSN.2011.6013894.
- Yizhong Wu. Development of hybrid modeling platform for multi-domain physical system[J]. *Journal of Computer-Aided Design & Computer Graphics*, 2006, 18(1):120-124.
- Xuan F. Zha, H. Du. Knowledge intensive collaborative design modeling and support Part I: Review, distributed models and framework. *Computer in Industry*, 57(1): 39-55, 2006. doi: 10.1016/j.compind.2005.04.007.
- Yanshan Zhang, et al. A knowledge-based web platform for collaborative physical system modeling and simulation[J]. *Computer Applications in Engineering Education*, 2013. doi: 10.1002/cae.21572
- Jianjun Zhao, Zijun Wu. Multi-domain modeling and co-simulation based on Modelica[J]. *Computer Aided Engineering*, 2011.

Control Development and Modeling for Flexible DC Grids in Modelica

Andreas Olenmark¹ Jens Sloth² Anna Johnsson³ Carl Wilhelmsson³ Jörgen Svensson⁴

¹One Nordic AB, Sweden, andreas olenmark@one-nordic.se.

²Gothia Power, Sweden, jens.sloth@gohiapower.com

³Modelon AB, Sweden, carl.wilhelmsson@modelon.com

⁴Industrial Electrical Engineering and Automation (IEA), Lund University, Sweden, jorgen.svensson@iea.lth.se

Abstract

This article presents a way of implementing different control strategies for power electronic converters in the Modelica modeling language. The different control modes were fitted into flexible models that could be interconnected in various power grid topologies. The power grid examples were controlled and kept stable during various load scenarios, using the developed controlled converter models. The work was performed using the Modelica tool Dymola. Modelica is an equation-based object-oriented modeling language. Electrical components in the Electric power library (EPL) from Modelon were used to model power electronic units, grids and other electrical infrastructures. The outcome of this effort was simulation results which clearly demonstrate that the developed controllers enable scalable and controllable DC power grids.

Keywords: HVDC, smart grids, converter control.

1 INTRODUCTION

The need for electrical energy is ever increasing and an increasing amount of the consumed energy is generated by renewable energy sources, even local and small power generators. In practice this means that the power distribution networks needs to be able to handle more power flux scenarios than they used to. This applies to electrical grids of all sizes whether it is a large national grid or a smaller household grid. The term “Smart Grids” is commonly used for these types of more flexible power grids. Common for these grid setups are that several units are coupled together and they need to be intelligently controlled to be able to keep the grid operational in the desired voltage level. The control units are power electronic devices e.g. converters

which can operate either on alternating current (AC) or in direct current (DC). These units are fast enough so that they supply sufficient grid controllability. The converter needs to be able to transform AC from an AC power source to a DC grid or consumer, and vice versa. It also needs the ability to control power flow, voltage, current etc. This work accounts for a simulation model of flexible size which implements these devices. By modeling these converters and grids, different applications where these grids are used can be designed, simulated and studied.

2 CONVERTER CONTROL

Power electronic converters are used for many different applications such as AC/DC transformation, DC/DC conversion, electrical drives, etc. Semiconductor based converters features high control bandwidth and high energy conversion efficiency and is therefore an important component in modern electrical grids.

2.1 THREE PHASE CONVERTER

A three phase converter consists of six transistors (most commonly IGBT transistors), in parallel with each transistor there is a freewheeling diode connected (see Figure 2.1). The converter can be divided into three transistor half-bridges where each can be viewed as a switch. The diodes are needed to provide a path for the inductor currents when the transistors switch off. Pulse Width Modulation (PWM) is used to form a switch pattern which in turn creates a controllable output voltage. It is hence possible to control e.g. specific machines, eliminate harmonics when connected to the utility grid or uninterruptable power supplies (UPS).

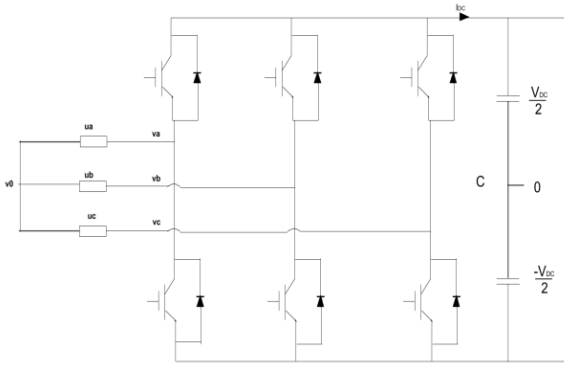


Figure 2.1 The electric schematic of the three phase converter connected to a load.

2.2 ACTIVE FRONT END (3-PHASE CONVERTER)

Usually, power electronic devices are supplied with DC voltage, conventionally obtained by rectifying the grid voltage using a diode or a thyristor rectifier bridge. The drawback with these rectifiers is that they introduce harmonic content in the current drawn from the grid and also lower the power factor due to its nonlinear properties. This result in a decrease in the power transferred across the line.

An active front end (AFE) is a three phase converter connected to the utility grid on one side and a DC grid on the other (see Figure 2.2). It has an inductance placed between the converter and the grid to provide voltage boosting, as well as to filter the currents. Capacitors are placed on the DC side to provide energy storage and to smooth the DC voltage. An AFE can be seen as a 4 quadrant DC/DC Boost Converter meaning that it can boost up the DC voltage to desired level and that it allows for power flux in both directions through the AFE (Carlsson, 1998).

By controlling the switching of the transistors the currents passing through the inductors are altered, hence the output DC voltage can be controlled. At the same time as the currents can be modulated to eliminate harmonics and to keep the power factor equal to one.

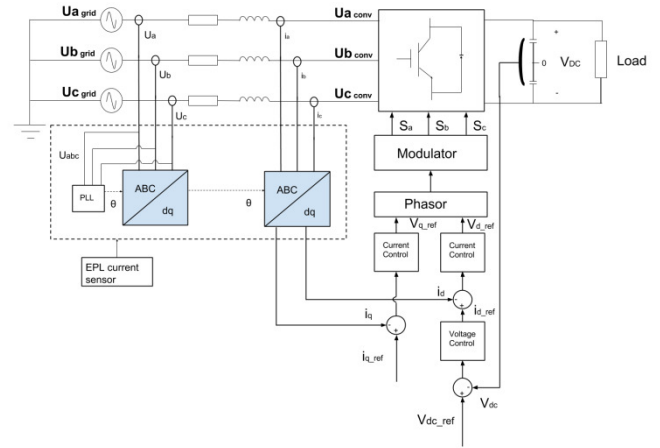


Figure 2.2 The AFE connected to the utility grid with DC voltage control.

The three-phase currents are controlled by applying the Park transformation to the currents, converting them into *dq* coordinates. The active and reactive power can then be expressed in terms of the grid voltage and current and can be controlled separately.

$$\begin{cases} P = Re\{e_{dq} \cdot i_{dq}^*\} = e_d i_d + e_q i_q \\ Q = Im\{e_{dq} \cdot i_{dq}^*\} = e_q i_d - e_d i_q \end{cases} \quad (3)$$

Since the grid voltage is aligned with the d-axis it means that $e_q = 0$ and by controlling i_q to zero the reactive power becomes zero.

$$P = Re\{S\} = e_d i_d \quad (4)$$

$$Q = Im\{S\} = 0 \quad (5)$$

The power factor is defined as $\cos \theta$ and unity power factor occurs when $\cos \theta = 1$ since:

$$\theta = \arctan\left(\frac{Q}{P}\right) \quad (6)$$

$$Q = 0 \rightarrow \theta = 0 \rightarrow \cos(0) = 1 \quad (7)$$

Thus by controlling the *dq* current separately unity power factor can be achieved. (Sanjuan, 2010)

2.3 POWER CONTROLLED AC/DC CONVERTER

Power can also be controlled using a three phase converter, since the transformed currents affect the active and reactive power. Using a three-phase converter with a structure as in Figure 2.3, the active and reactive power can be controlled.

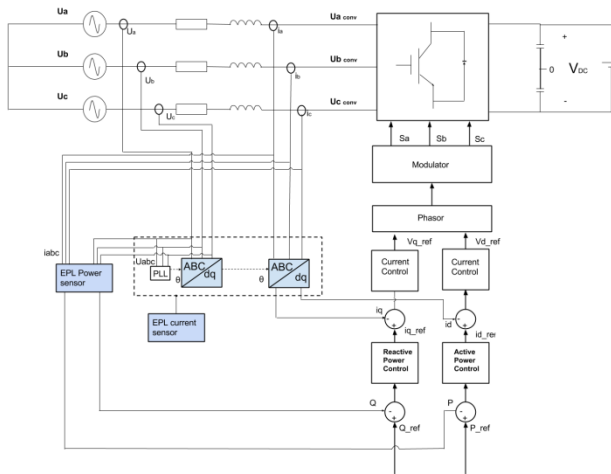


Figure 2.3 an illustration of the implemented power controller. The control logic implemented is a cascade controller, the outer loop controls power and the inner loop current.

2.4 DROOP CONTROLLED AC/DC CONVERTER

A combination of power and voltage control is called droop control. Droop control is in this work used to ensure voltage stability of the internal DC grid if, for some reason, the primary voltage source is disconnected or malfunctioning. The errors of the power and voltage are added together and the quantity which deviates most from its reference is subject of control, this can be seen in equation (8). By adding weights to the errors the droop control is achieved. The setup for this control strategy is shown in Figure 2.4.

$$e_{p-controller} = e_{V_{DC}} \cdot k_1 + e_P \cdot k_2 \quad (8)$$

Where

$$\begin{cases} e_{V_{DC}} = V_{DC,ref} - V_{DC} \\ e_P = P_{ref} - P \end{cases}$$

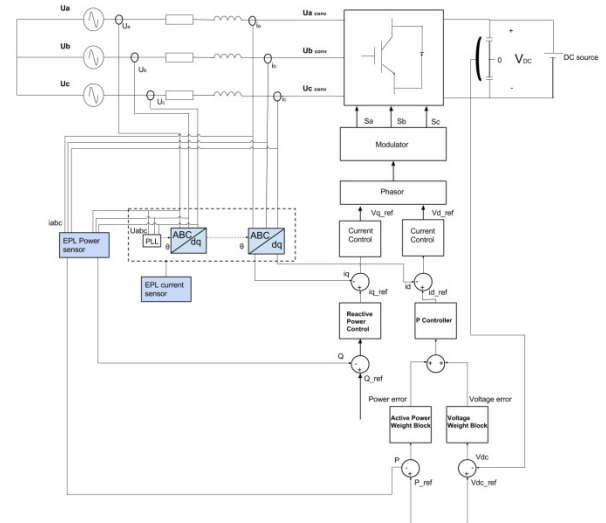


Figure 2.4 droop control structure.

3 EXPERIMENTAL SETUP

The models of the different kinds of power electronic converters were put together using Modelica and Dymola. The control logic of the converter models were made using infrastructure from the Modelica standard library (MSL). In order for the converter models to be flexible, to operate in different control modes and at different power levels, a converter component with interchangeable control schemes was constructed. The flexible-control converter offers the possibility to choose between voltage, power or droop control. Depending on which type of grid that needs to be simulated, different power and voltage levels need to be selected in the component. The flexible-control converter model can easily be set up and parametrized using records.

3.1 GRID SCENARIO SETUP

To evaluate the converter and grid models, different scenarios were developed. E.g. a DC grid consisting of several converters connected in parallel (Multi Terminal Direct Current, MTDC), and nets with only two converters connected together. The converters were always connected with each other Back-to-Back, regardless of the amount of devices. In every system there had to be at least one converter responsible for keeping a constant DC voltage level on the DC grid. The one controlling the DC voltage is called master terminal and all other terminals are called slave terminals. The slave terminals were responsible for controlling power to the different sub units. The master terminal was the

one maintaining power flow balance in the DC grid (Haileselassie, 2012).

3.2 MULTI-TERMINAL HVDC SCENARIO

The first grid examined was a MTDC grid for High Voltage Direct Current (HVDC). In this high voltage scenario example four multi controlled converters were connected in parallel and an AFE controlling the DC voltage was connected to the utility grid. The DC voltage level was set to be 130 kV. The converter components, intended to be connected to different generators, e.g. hydro-power plants or wind-power plants, were set to perform power control. The generators and the utility grid were represented by AC sources with 50 kV line-line voltage, operating at 50 Hz. The total maximum power of the generators was assumed to be 400 MW, 250 MW for the two wind-power plants and 150 MW for the hydro-power plant. The wind power plants were set to generate full power initially and then vary its output slowly according to a low frequency sinusoidal reference. The output of the larger of the two wind-power plants varied between 80-150 MW, with a frequency of 0.01 Hz. The output of the smaller wind-power plant varied between 60-100 MW with frequency 0.05 Hz. The hydro-power plant generated full power and was shut down periodically. This behavior was achieved by using square wave as input to the power reference of the converter connected to it. The Modelica implementation of the grid setup can be found in Figure 4.1.

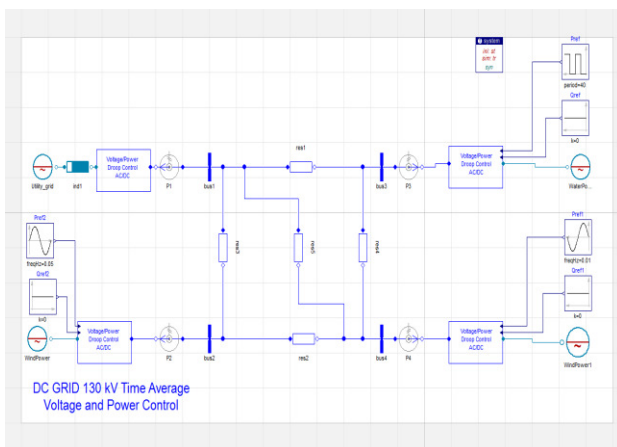


Figure 3.1 Modelica implementation of the HVDC grid

3.3 HOUSEHOLD WITH LOCAL POWER PRODUCTION

The second grid topology which was studied was developed to mimic a typical household with local

power production. Two wind turbines were connected to the DC grid as well as a model of a “smart” house. The wind turbines had capacity of 4 kW each. The AC distribution grid was connected to the DC grid through an AC/DC converter. The converter was set to control DC voltage to 1.5 kV. The AC distribution grid was represented by an AC source operating at 50Hz and 400 V line-line. Additionally a solar power plant, represented by a DC source with voltage 100 V, was connected to the DC grid through a power controlled DC/DC converter. The power “generated” by the solar plant varied according to a sinusoidal between 0-2 kW. The Dymola implementation of the grid setup can be found in Figure 4.2.

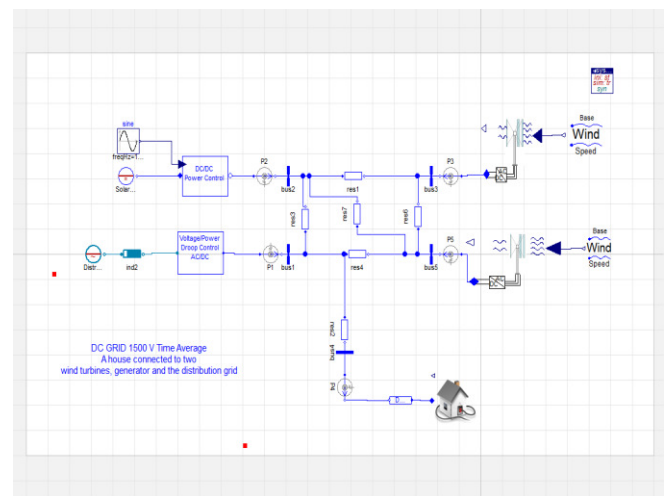


Figure 3.2 The grid setup for the DC grid with Smart Grid Library and Wind Power Library models connected

4 RESULTS

4.1 MULTI-TERMINAL HVDC SCENARIO

The power-controlled AC/DC converters should deliver the power generated by power sources to the internal DC grid. The voltage at the internal DC-grid/utility-grid connection point was to be kept constant. The power generated by the power plants was to be fed to the utility grid.

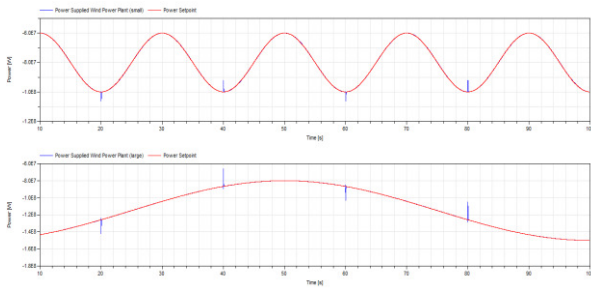


Figure 4.3 The power generated by the wind power plants fed to the internal DC grid, 250MW (top) and 400 MW (bottom).

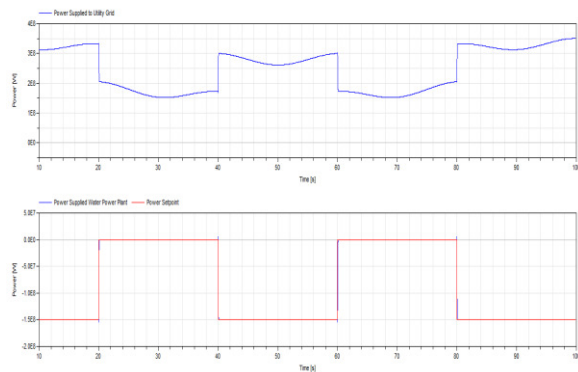


Figure 4.4 The power generated by the hydro power plant (bottom) and the power supplied to the utility grid (top).

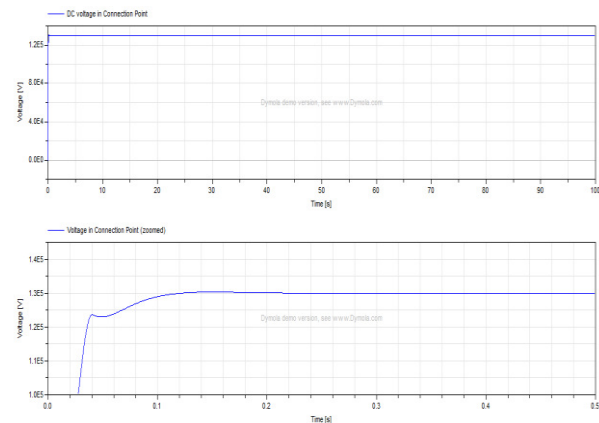


Figure 4.5 the voltage at the connection point of the DC grid to the utility grid (zoomed in bottom).

The power which was generated by “wind-power” and fed to the internal DC grid can be found in Figure 4.3. Figure 4.4 shows corresponding for the hydro power plant. The limiting factor was the speed of the power controller as visible in Figure 4.4, where a large change in power control reference value takes place momentarily. The power controller responded very quickly to this change and it hence proved adequate. Sudden changes which were made to the hydro power plant reference proved to affect

the wind power plants. Sudden changes in power will of course affect the current flowing through the circuit and thus the power which had to be supplied by the power controlled converter connected to the wind power plants. Even though there was an undershoot occurring the recovery was swift and the system maintained stability.

The voltage in the connection point between the utility and internal DC grid was kept throughout the simulation and was not affected considerably by the rapid changes in power flow as can be seen in Figure 4.5.

The generated power was supplied to the utility grid as visible in Figure 4.4.

Since the resistance of transmission lines is large for longer lines, a considerable loss in power will occur while putting current through long transmission lines. The losses can be minimized by raising the voltage of the DC grid, but this will put a larger demand on the power electronic equipment and also add to the investment cost of this equipment (Alaküla, 2011).

To keep the power balance the voltage of the internal DC grid should be kept constant when changing the load characteristics and with varying power supply. The power supplied should be consumed by the load when load is active. The remaining power should be fed to the AC distribution grid. On the other hand, the AC distribution grid supplies power when the local generation is insufficient.

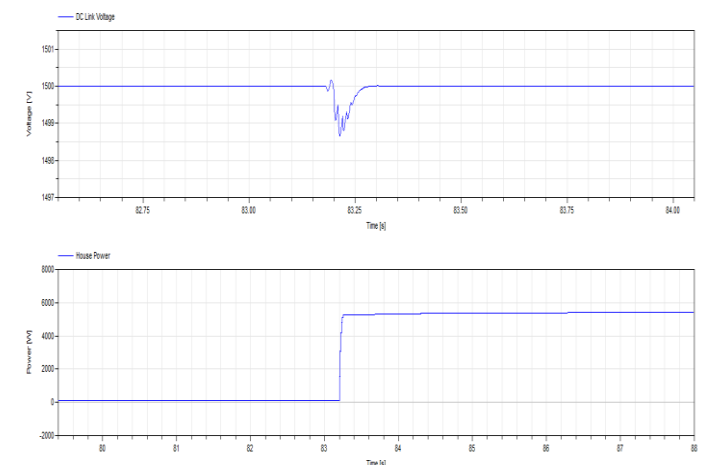


Figure 4.1 The voltage (top) of the DC grid reacting on a sudden change in power consumption (bottom) in the load.

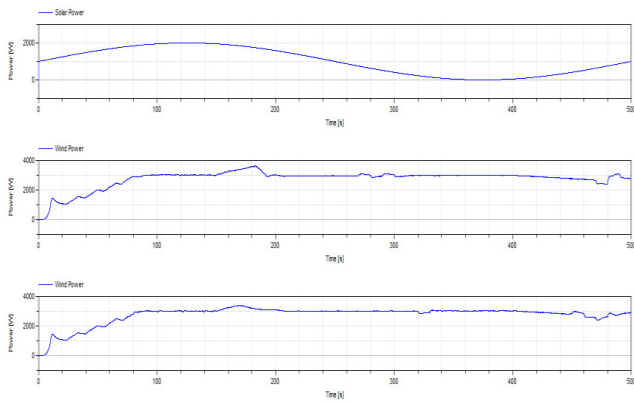


Figure 4.2 The power supplied by the generators, in the wind power plants and for the solar power plant (top)

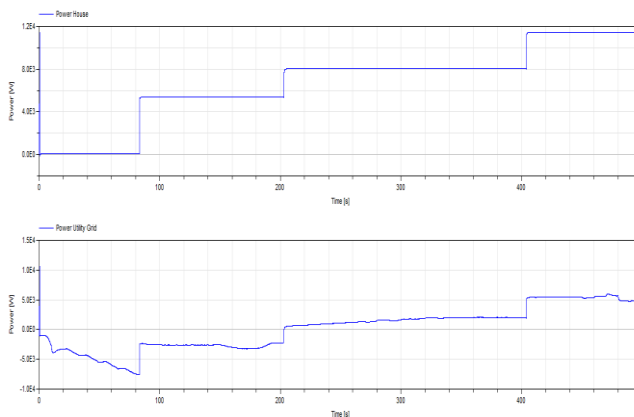


Figure 4.3 The power consumed by the load (top) and the power supplied to/consumed from the AC distribution grid

4.2 HOUSEHOLD WITH LOCAL ENERGY PRODUCTION

The voltage of the DC grid could be kept at 1.5 kV as can be seen in Figure 4.6. When the load was changed the voltage of the DC grid undershoots, it was however quickly brought back to the set point of 1.5 kV by the voltage controller. The largest part of the power that was consumed by the load was generated by the local wind and solar power plants, see Figure 4.7 and 4.8. The excess power was fed to the AC distribution grid as can be seen in Figure 4.8. In the same figure it is also shown that the AC distribution grid supplies the remaining power consumed by the load when local production units were unable to keep power balance.

5 CONCLUSIONS

The main purpose with the work undertaken was to implement a scalable, flexible and controllable grid. The term scalable implies that the grid could have

been used in many different voltage and power levels, meaning that the grid could represent different applications.

As seen from the results the converters and the implemented control strategies maintains grid voltage and power flow as expected. This shows that the models can be interconnected and works in a rather stressed environment. It also shows that an arbitrary grid can be constructed and simulated using these flexible models for system and grid analysis. The two different grid models show the flexibility of the constructed examples. In addition the constructed models were tested together with external library models such as wind power models and energy consumers with local generation. This demonstrates compatibility with other libraries.

The compatibility with other libraries enables a further development of the grid models. In this article two examples are demonstrated but these models provides the opportunity to simulate other scenarios. For example internal grids in vessels, charging stations for electric vehicles, internal grids in vehicles or the internal grid of several wind power plants can be simulated using these flexible models.

The opportunity is given to implement more efficient systems and to evaluate new ideas. The models enable integration of new technologies for renewable power generation and efficient distribution of power. These models provide the opportunity to simulate and, in extension, realize smart grid implementations.

Neither switching nor thermal losses were taken into account here. In order to get more accurate and realistic results losses should be included in the analysis. The EPL components used in the construction of the models support these kinds of losses, enabling these effects is however regarded as future work.

NOMENCLATURE

\vec{e}_d - *d*-component of the grid voltage

\vec{e}_q - *q*-component of the grid voltage

\vec{e}_{dq} - Grid voltage in *dq* coordinates

U_{dc} - DC link voltage of the three phase converter

v_a - Output voltage of the three phase converter phase a

v_b - Output voltage of the three phase converter phase b

v_c - Output voltage of the three phase converter phase c

v_0 - Zero potential of the three phase converter

u_α - Phase voltage expressed in the real component of the complex α, β vector.

u_β - Phase voltage expressed in the imaginary component of the complex α, β vector.

u_a - Voltage over the load of the three phase converter output phase a

u_b - Voltage over the load of the three phase converter output phase b

u_c - Voltage over the load of the three phase converter output phase c

$|\vec{x}|$ - Magnitude of the maximum output voltage vector of the three phase converter

i_d - d-component of the dq current vector

i_q - q-component of the dq current vector

REFERENCES

M Alaküla, P Karlsson, "Power Electronics: Devices, Converter, Control and Applications", *Department of Industrial Electrical Engineering and Automation*.

M. Alaküla, L. Gertmar, O. Samuelsson, "Energiteknik", *Lund, Sweden, Lunds Tekniska Högskola, Department of Industrial Electrical Engineering and Automation, 2011*

A. Carlsson, "The back to back converter control and design", *Lund, Sweden, Department of Industrial Electrical Engineering and Automation Lund Institute of Technology, May 1998*.

T. M.Haileselassie; "Control, Dynamics and Operation of Multi-terminal VSC-HVDC Transmission Systems", *Norwegian University of Science and Technology Department of Electric Power Engineering Trondheim December 2012*.

S. L. Sanjuan, "Voltage Oriented Control of Three-Phase Boost PWM Converters Design, simulation and implementation of a 3-phase boost battery charger". *Department of Energy and Environment, Division of Electric Power Engineering CHALMERS UNIVERSITY OF TECHNOLOGY, Göteborg, Sweden, 2010*.

Towards Enhanced Process and Tools for Aircraft Systems Assessments during very Early Design Phase

Eric Thomas¹ Olivier Thomas¹ Raphael Bianconi¹ Matthieu Crespo² Julien Daumas²

¹Dassault Aviation, France, {eric.thomas, olivier.thomas, raphael.bianconi}@dassault-aviation.com,

²Liebherr Aerospace, France, {matthieu.crespo, julien.daumas}@liebherr.com

Abstract

This paper deals with an improved process for early to detailed design phases of complex Aircraft systems. It is based on experience of Dassault Aviation (DASSAV) and Liebherr Aerospace Toulouse (LTS) in aircraft system design, and on works carried out within several R&D projects, in particular within current FP7 TOICA project (Thermal Overall Integrated Conception of Aircraft), where new process are developed to tackle assessments of architectures composed of many heterogeneous and interconnected sub-systems using simulation. This new process that will be described in this paper involves open standards like Modelica and FMI.

Keywords: Collaborative process, System engineering, MBSE, multi-levels simulation, PLM/SLM integration

1 Introduction

Aircraft vehicle systems are typical examples of complex systems, composed of many sub-systems provided by several companies, which overall represent a set of thousands of equipments with many interactions between them. These systems must meet numerous performance and safety requirements.

Architectures trade-offs require different kinds of analysis, in particular behavioural assessments. The purpose of this paper is to define vehicle system architectures, investigate the current performance assessment process and propose an improved process based on models exchanges and simulations, applicable during preliminary design phases like RFI (Request For Information) or RFP (Request for Proposal):

The article is structured as follows:

- Section 2 briefly presents aircraft vehicle systems architecture and their representations.
- Section 3 analyses the current design process.
- Section 4 explains current issues in air system design and solutions developed within the project FP7 TOICA
- Section 5 presents solution used within the project and the challenges ahead to get a full and efficient set of tools and processes for future airplane designs

The analysis will start by the description of the current process for architectures assessments, by defining:

- The content and representation of an architecture
- The required activities to assess architectures during the different design phases.

2 Aircraft architectures

The Aircraft systems architectures are generally managed using tree views (product break-down ...) and 2D views as represented in fig.1. They allow to describe the system by hierarchical decomposition within different levels (e.g. aircrafts, systems, sub-systems ... components/devices ...) represented here as boxes with connections between them (Internal Block Diagram according to SysML).

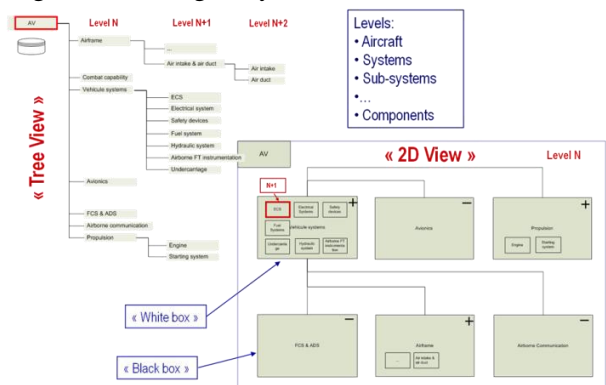


Figure 1. Typical Aircraft (A/C) Logical Architecture

This global system/product is developed with many partners for which access rights to the whole aircraft definition, and associated models, can be restricted and depend on roles of users (e.g. Dassault Aviation Architects, Designers, partner designers ...) They have only access to items they are responsible and, within limits, to the borders and particular information of other surrounding systems. So several systems could appear for users as “white boxes”, with complete access to all information, “black boxes” or as “grey boxes”, with only access to several published information according to user’s role. And this status could vary along the time and the design phases.

These representations are very useful to describe the global architecture and for the navigation within it. For example, if the “vehicle system” node is selected in the tree view, it is possible to get more information on elements of this layer and to focus on sub parts to get more detailed information, as represented in the following figure 2.

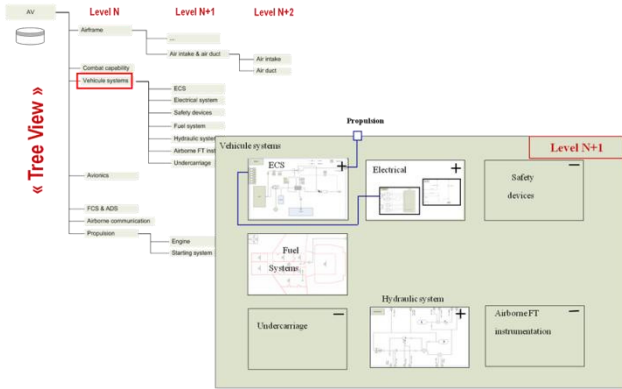


Figure 2. Typical Vehicle Systems Logical Architecture

In addition, for design specialists, elements of the systems could be represented using standards (e.g. symbols) or with not standard representations (e.g. product images ...) to quickly identify main functions of components. In this way, several typical sub-systems are represented below with different technical representations and layout. And, according to the previous hierarchical representation, their below to different levels, here layers N, N+1 et N+2.

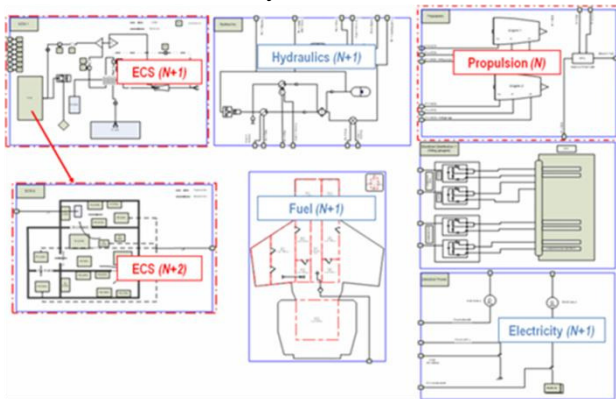


Figure 3. Sub-systems and associated components at different levels (very simplified views)

In fact, representations of complex systems with multiple viewpoints already exist and we can find associated tools to manage them in association with the digital Mock-Up (DMU) to manage them in collaborative context with IP (Intellectual Properties).

But, because of the large amount of requirements at aircraft level and attached to each sub-system, assessments and trade-offs between such architectures are not easy to perform in a flexible and efficient way. Furthermore there are currently no assessment means adapted for complex architectures design in particular during preliminary phase.

Therefore, new innovative processes and associated tools are studied and developed within the project FP7 TOICA. For this, Dassault Aviation and Liebherr Aerospace Toulouse have chosen to rely, as much as possible, on standards (Modelica, FMI, SysML ...). The following paragraphs will detail analysis needs, and process and tools developed using particularly Modelica and FMI.

3 Analysis of the global design process

To really understand the benefits of the new process improvements, the current design process, workflows and involved actors will be explained.

3.1 Current Design process workflow

The figure below (fig.4) tries to illustrate workflows and traceability links between the different tasks carried out during the design of the vehicle systems by Aircraft systems integrator.

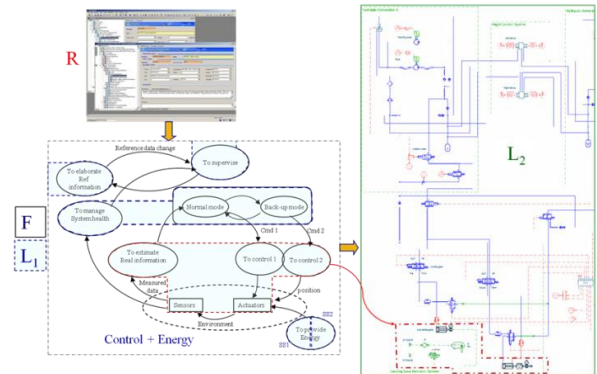


Figure 4. Design process workflow

From the top level requirements (R) are established the aircraft functional analysis which define functions (F) to be performed by the system, and that will be fulfilled by parts of it, sub-systems and components (e.i. equipments ...). These functions are then grouped together to be assigned to sub-systems (L1 view in fig.4), and then to partners as packages called "Product Packages". Partners will then provide solutions implementing the required functions (L2 view).

3.1.1 Design phases

The works illustrated above vary along the time from Preliminary system design to detailed Component development. They are developed within next paragraphs regarding design phases.

3.1.2 Preliminary Design

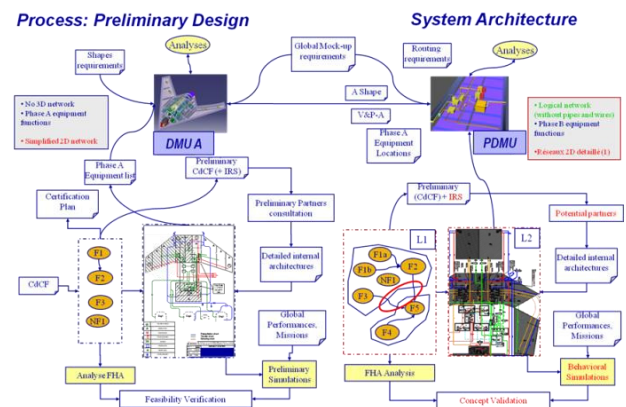


Figure 5. Process regarding Preliminary Design and System Architecture phases (1/2)

As illustrated above (fig.5), during the Preliminary Design phase of a new Aircraft, a simplified DMU (Digital Mock Up), often parametric, is set-up. And from the preliminary functional requirements, the main functions are defined.

At that point, first global architectures are defined and evaluated. For this, consultation of potential partners is made during the RFI (Request For Information) and RFP (Request For Proposal) phases.

For the preliminary global architecture assessments, different activities can be carried out:

- From the functions, it is possible to make preliminary analysis such as a FHA (Functional Hazard Analysis)
- From the logical architecture, overall performance, missions, operational scenarios, it is possible to make behavioral assessments, reliability and availability estimates. This is currently based on Dassault Aviation models and a few available information provided by partners that are not yet been selected. In this evaluation phase, there will be a clear advantage of being able to use more detailed models from potential partners.
- Other analyses are made in parallel, in particular to define the aerodynamic shape of the aircraft.

3.1.3 System architecture selection

During system architecture selection, a preliminary DMU is used to make several types of analysis like engine burst analysis or space allocation.

Functions are refined and preliminary interfaces are assigned among potential Product Packages through ICD (Interface Control Documents).

3.1.4 Component specification

During Development Phase, partners have been selected. It is now possible to ask them detailed information on their solution(s), and models to be evaluated in a more global and multi-systems context.

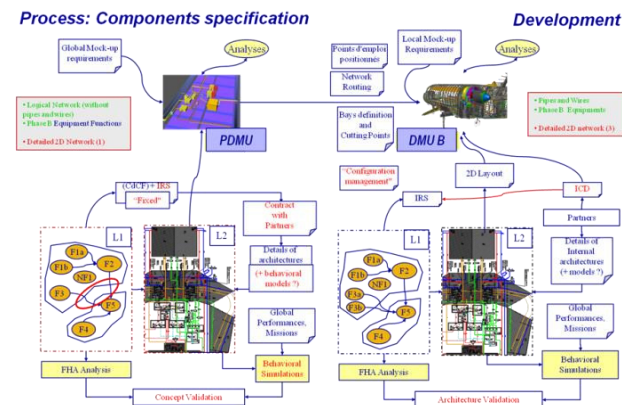


Figure 6. Process regarding Preliminary Design and System Architecture phases (2/2)

3.1.5 Development

During development phases a detailed DMU is set-up and becomes the main reference to build the aircraft. It is fixed when all detailed sub-systems are defined with sufficient details and all Critical Design Reviews passed.

3.2 Actors

Many actors take part to the aircraft design. They are in charge of different roles and are allowed to see and interact on subsets of the whole aircraft definition. Figure 7 shows some of the actors involved in TOICA project.

Actors	Name	Role : In charge of
Aircraft architect	DASSAV	Aircraft Architecture : - Launch trade-off - Lead Collaborative Design Reviews for the whole Aircraft
Multi-Systems Architect	DASSAV	Multi-Systems (Energy) Architecture : - Definition , validation of Energy systems
Thermal Experts	DASSAV	Air-Conditioning systems : - Definition, development and justification
Anti-Icing Experts	DASSAV	Anti-Icing system : - Definition, development and justification; in partnership with Thermal Experts
Process And Tools Expert(s)	DASSAV	Process and Tools for Modeling & simulations : - Provide Tools and Process for M&S - Manage models repository
CAE Analyst(s)	DASSAV	CAE expert (CFD, FEA) : - CFD calculations - Models reductions
Partners Thermal Experts	LTS	Partner Representative : - Process and Tools Expert / Thermal Expert - Provide sub-systems models

Figure 7. Actors of Dassault Aviation-LTS use case

3.3 Managing alternatives

During early phases, there is a need to manage alternative architectures. Starting from a high level Functional representation, sub-level functions can be grouped together in different ways. Architects and design experts will select different candidate architecture according to overall needs, and solutions provided by potential partners.

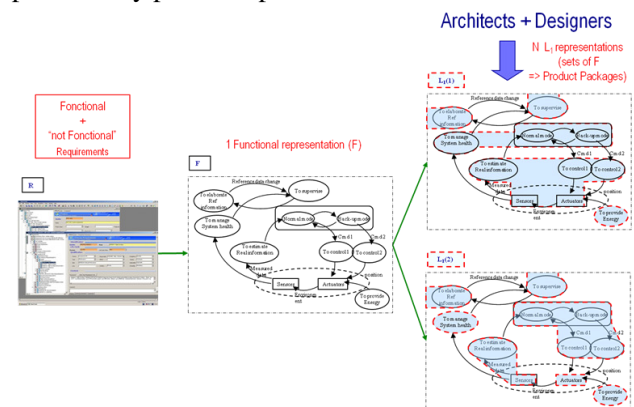


Figure 8. Process and management of alternatives

There are thus several possible Functional representations during early phases which gather sets of functions. But functions don't implement all the requirements. The Non Functional requirements like constraints must not be forgotten, and they must be allocated to the sets of functional requirements as illustrated in fig.9 below.

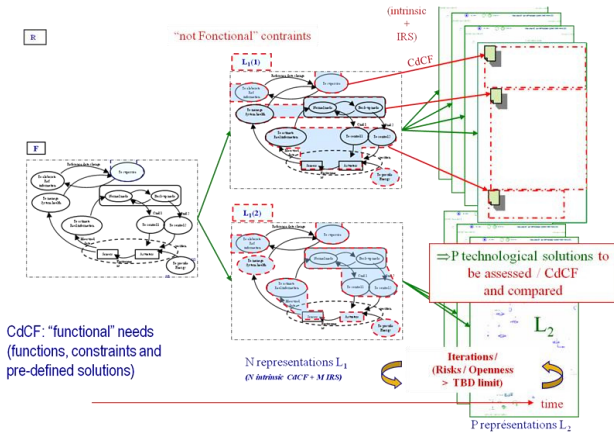


Figure 9. Process and analyses for architecture assessments

Note that openness on sub-systems requirements left to partners should allow them to propose more innovative solutions. But the larger the aperture is, the greater the industrial or contractual risk may be.

3.4 Multi-systems simulation

In TOICA, one of the purposes is to investigate more formal exchange by models to allow assessments not only limited to sub-system, but for an overall analysis at multi-systems level (air system, including other boundary systems) and at aircraft level.

For this, models involved in DASSAV-LTS TOICA use-case are defined just below.

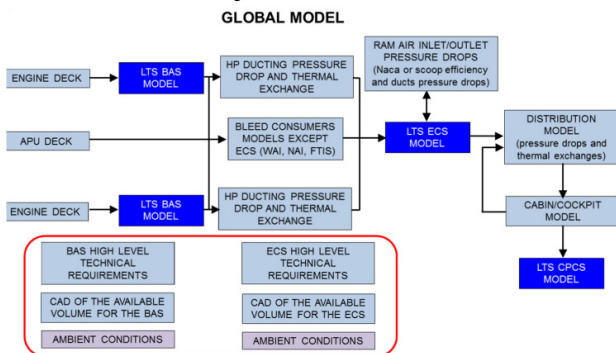


Figure 10. RFI phase models

We can note that partner’s subsystems can be highly scattered among the whole system.

The associated architecture could be also defined as below in a SysML (OMG) format:

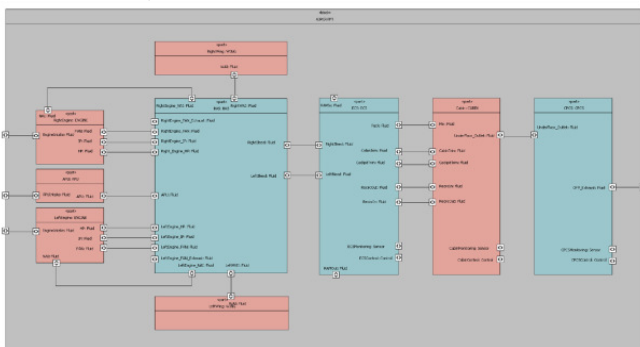


Figure 11. IBD with LTS systems in blue

4 Enhanced process definition

4.1 Introduction

4.1.1 Current process

To be able to make multi-systems assessment, the aircraft systems integrators generally ask for models to system designers. But when trying to build the global simulation, problems often occur when connecting or during simulation of the coupled models. The interfaces don’t match as wished, simulations are much slower as expected, models don’t publish all expected variables... according to real needs that are not all known at the beginning of the system development ... The process requires more efficiency and flexibility.

4.1.2 New process

The primary purpose is to give to the potential partners the capability to check the ability of their models to run efficiently in the Dassault Aviation simulation framework. An additional target is to allow quicker design iterations between Dassault Aviation and partners to test more solutions and potentially more innovative ones.

In TOICA, the foreseen solution is then to provide to potential partners a simulation framework allowing them to carry out previous tasks. This new process of exchange and use of interfaces values and requirements by models in a common framework will ensure a better flexibility, efficiency and traceability.

The Dassault Aviation framework and rules to use it are detailed in next paragraphs.

4.2 Analyze of current design process.

The current process between Aircraft manufacturer (Dassault Aviation) and partners (here LTS) at RFI / RFP phase is analyzed. It is illustrated in the following figures, as the workflow from the technical specification provided by Dassault Aviation to partner selection, and for two activities:

- Modification of technical specifications
- Modification of a sub-system such as an ECS (Environmental Control System) pack

4.2.1 Modification of Technical specification

The workflow is defined in fig 12. If Aircraft manufacturer wants to modify one or several technical requirements, the new issue must be send to the partners to calculate the impacts on their systems.

Partner’s results have to be sent to the Aircraft manufacturer. Then aircraft architects analyze impacts to decide:

- to request partners a system modification or redesign
- to modify the specification to decrease the impacts (e.g. a new calculation loop with partners is needed)

With this process, iterations between Aircraft manufacturer and partners are required for any modifications of specifications. It requires many sequential tasks which take times to fulfill all the technical specification items.

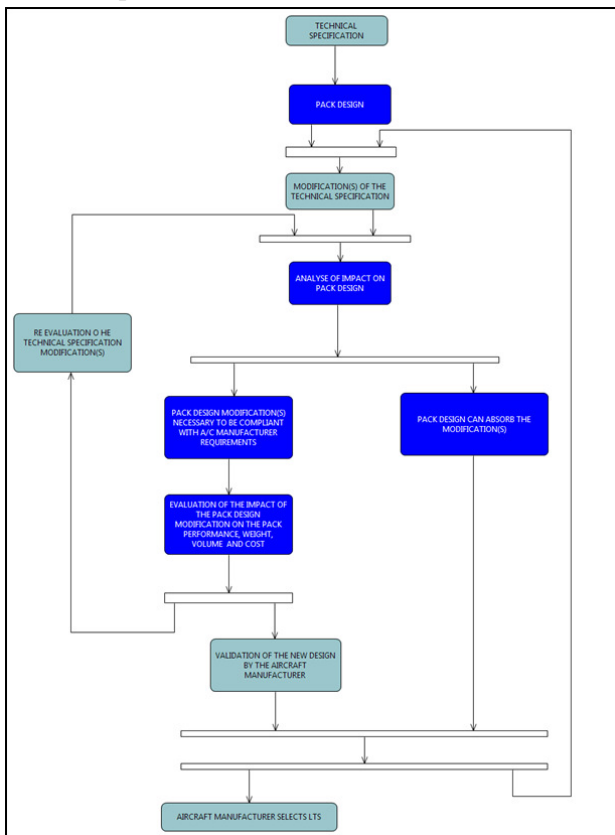


Figure 12. Current design workflow

4.3 Modified design process

The new workflow, sketched in fig.13, uses exchange of models. As it will be described in the following paragraphs, all needed models to make systems assessments will be available at Aircraft manufacturer and at other potential partner’s offices. Therefore, it allows them to work in parallel.

It enables Aircraft manufacturer to partially evaluate the impacts of these modifications on the partners systems.

Aircraft architects will be able to quickly analyze modification impacts and to decide:

- To request partners some system modifications or redesign
- To adapt the specification to improve the Aircraft without partners system modifications

Then, calculations results and specifications modifications will be sent to partners for validation.

Therefore iterations between Aircraft manufacturer and partners (LTS and other competitors) are reduced and the Aircraft architect can evaluate in a shorter time the impacts of the proposed modification. Aircraft architect will be able to evaluate and adapt the impacts of modifications without iterating with partners.

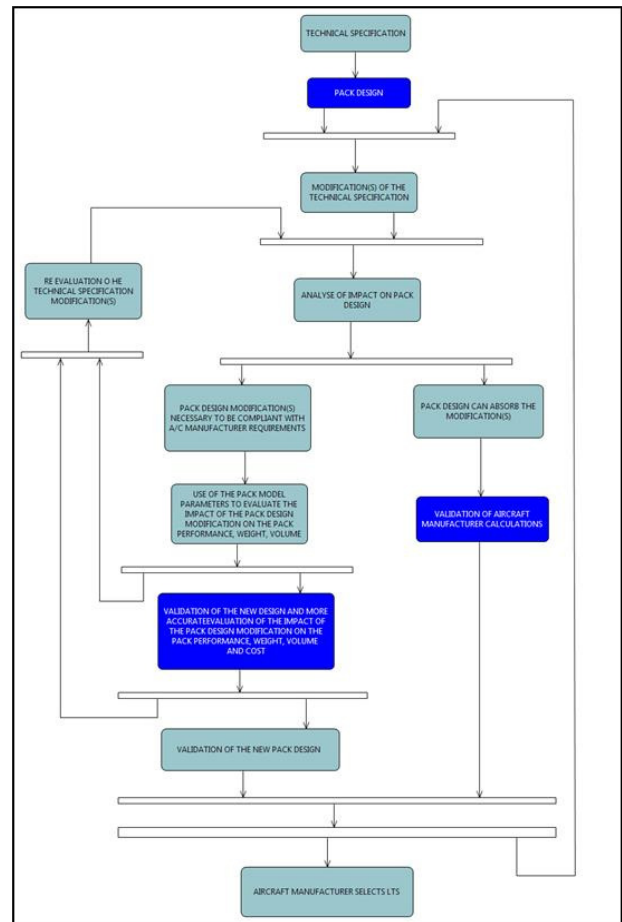


Figure 13. New process

Models exchanges enable partners to evaluate the impacts of system component modification on the aircraft before proposing it to Aircraft manufacturer.

The Integrated Air Management System (IAMS) engineer will be able to evaluate and adapt the impacts of IAMS components design without iterating with Aircraft integrator.

4.4 Modification of sub-system

For a sub-system design modification (i.e. ECS Pack) similar advantages could be pointed out as figured out in the following fig.14 (but not detailed in this paper).

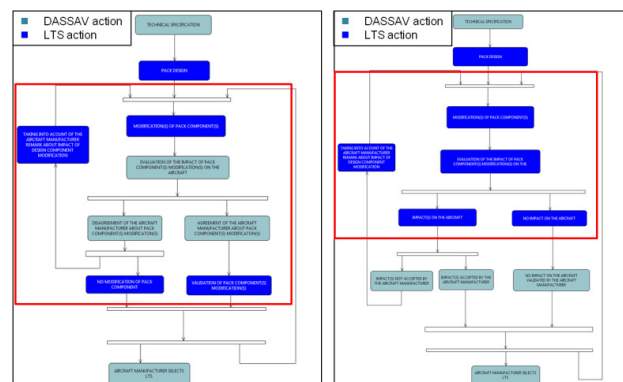


Figure 14. Comparison between the two process

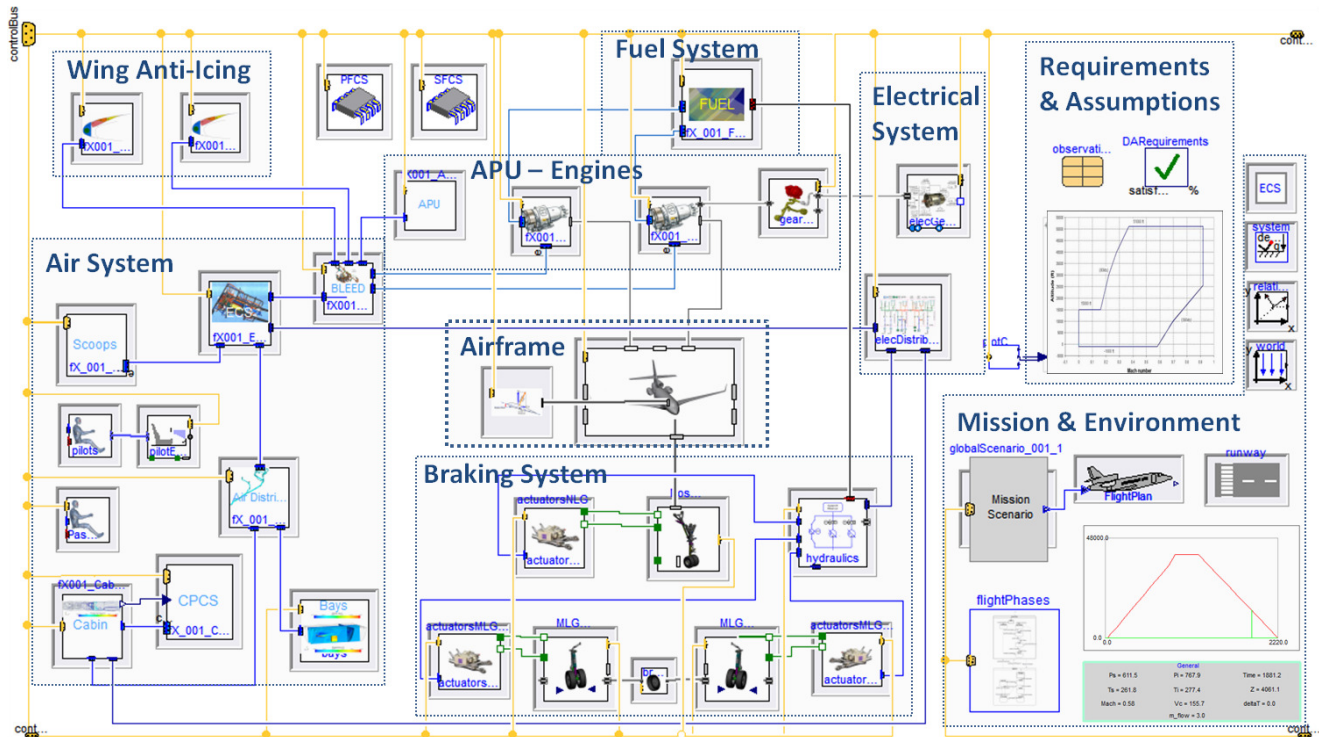


Figure 15. Flat representation of a set of inter-connected systems of a traditional Aircraft Vehicle systems architecture

5 Solution to set up the new process

The purpose is to allow an easy connections between models, checking of simulation capabilities as early as possible (within partner’s office), before final check and integration within Dassault Aviation global model. The current solution is to provide model Interfaces, boundary models, Mission and test cases to partners. The solution has been defined and implemented as an encrypted Modelica/Dymola library to protect IP (Intellectual Property). It contains the different elements described below:

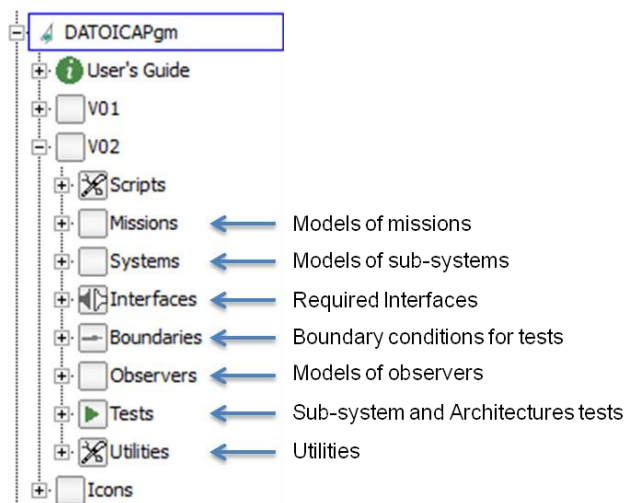


Figure 16. Library content

Note that before encryption, which is specific to Dymola, all components of the library, including protection annotations, are standard Modelica (currently according to Modelica Specification issue 3.3).

5.1 Components of the library

The library is mainly composed of models of missions, sub-systems and tests to allow architectures evaluations, and also interfaces to integrate partner’s models. Fig. 38, at the end of the paper, shows more details on the current library content.

The following paragraphs describe the major elements and the philosophy of the library.

5.1.1 Principles

The library takes advantage of principles used within the Modelica VehicleInterface library described particularly within (M. Dempsey *et al*, 2006). The VehicleInterface library, dedicated to automotive systems and architectures, was derived by Dassault Aviation for use with Aircraft Vehicle Systems during ITEA2 EUROSYSLIB project. A traditional Aircraft Vehicle systems architecture, extract from this library, is represented in fig.15. It represents the main inter-connected elements of such architecture as replaceable components, contrained by predefined interface. All components are also linked together with Simulation Control bus in charge of propagation of information

from the controllers and from the Mission and Environment models.

At the bottom right are environment components which contain global parameters and models that can be used by any other components using the Modelica inner/outer mechanisms. In these parts are represented the World and Fluid system components of the MSL, and other components dedicated to aircraft environment (FlightPlan, Runway, RelativeWorld ...) but also other components with global domain specific parameters and models (ECS, Braking system, Fuel system...) Note that contrary to the Vehicle Interface library where all connectors are single MSL components, the physical connectors have been replaced by composite connectors to limit the number of connections between sub-systems, and to allow a better flexibility when change of interface definitions.

5.1.2 Examples of elements: Mission Scenario

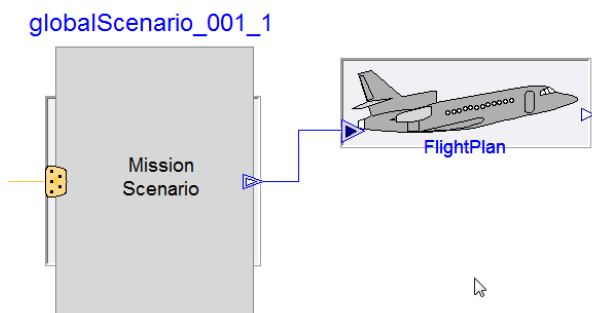


Figure 17. Mission and FlightPlan component icons

The Mission component is a key element of the global model. It provides all information about the predefined aircraft mission scenarios. It may define dynamic scenarios with some information varying along time as represented in the following figures.

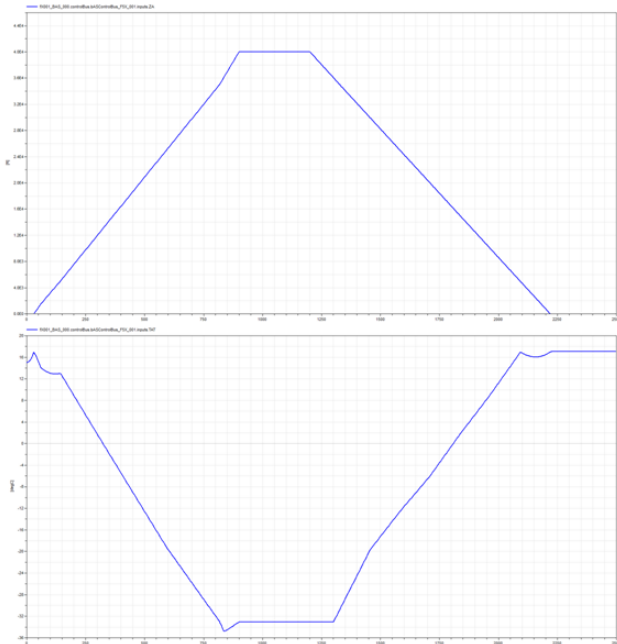


Figure 18. Plots of Aircraft altitude and external stagnation temperature

It is also simple to define stationary scenario if needed with constant outputs. Within the information provided by the Mission components, some information generally depend only on flight phases or are constant along time. There are then defined within the Mission component through graphical user interfaces such as represented in fig.19.

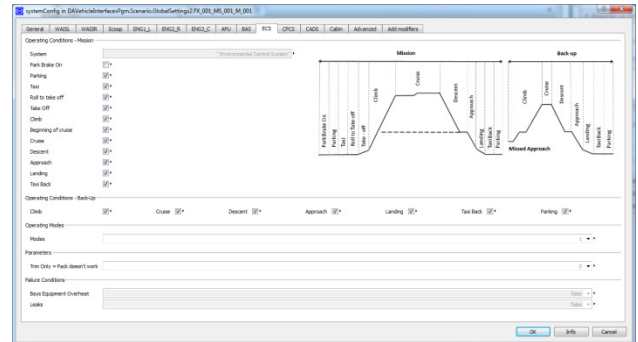


Figure 19. Configuration parameters for the ECS

The Mission model is connected to the FlightPlan component which provides all consistent information on Aircraft properties and associated external variables (altitude, attitude, velocity, Mach number, Temperatures, Pressures, Humidity ...) which must often be known by models of sub-systems.

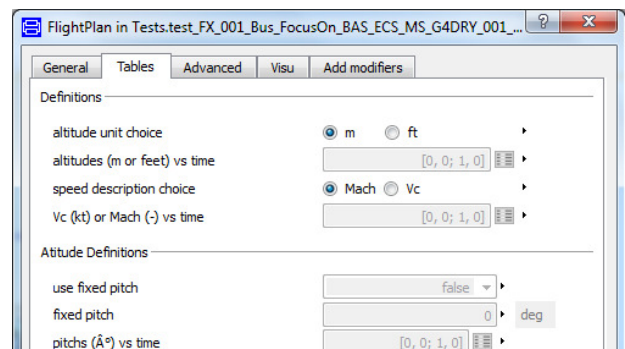


Figure 20. Part of GUI of the FlightPlan model

The mission model may recover information from sensors or from the different sub-systems models. Among them, Aircraft phases are modeled by a Modelica state diagram (Modelica synchronous), such as the one represented in the following figure.

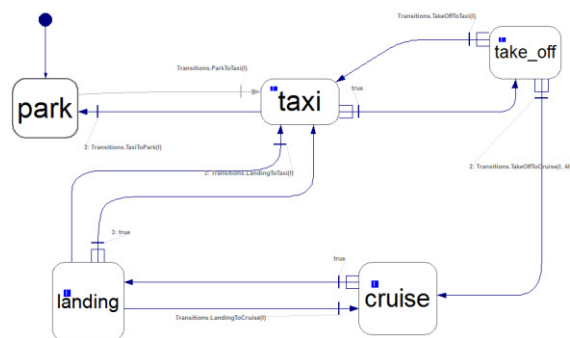


Figure 21. Simple Flight phases model

5.1.3 Examples of sub-system models: cabin

The cabin and cockpit may be modeled with different level of details. For the RFI/RFP phases, where still few detailed information are available, some simple models as represented below may be sufficient. In this model, crew and passengers (Pax) areas are represented by only big thermo-fluid volumes externally connected to other sub-systems.

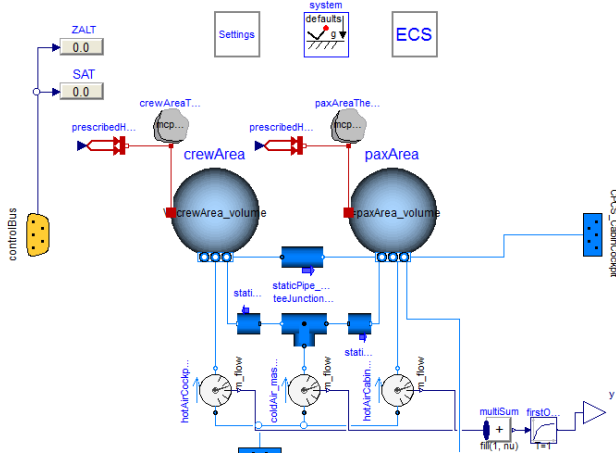


Figure 22. Simple model for Cockpit and Cabin

Heat exchanges are modeled by prescribed heat flows according to relation that here depends only on the number of crew members and passengers and the different other heat sources within the volumes and exchange with boundaries. These heat exchanges are here defined by parameters related to aircraft altitude (ZALT) and external static temperature (SAT). These two input parameters are provided by the Mission component through the simulation control bus.

Parameters can be set manually by users. Here below is presented the graphical user interface available for the partners, that allows them to change parameters if needed.

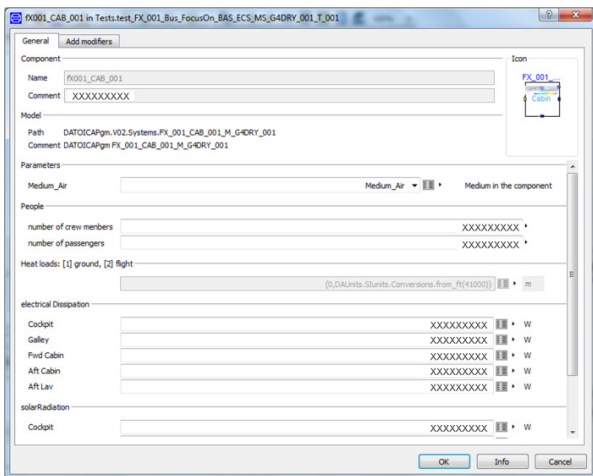


Figure 23. Cabin parameters available for Partners

In fact this component is also available at Aircraft integrator facility, with more published parameters.

5.2 Library usage

Currently, systems requirements for performance assessment, asked to partners, are provided as textual format.

One of the purposes of the library is to provide more formal requirements, as models, and tests that must be completed by partners to verify the compatibility of their solutions with requirements. They are the translation in models of current textual requirements.

Below are presented scenario and checks that should help checks of partner's systems.

5.2.1 Global scenarios assessment

Within the Tests packages, are provided tests of sub-systems and more global models for architecture assessments of partner sub-systems. In this case, the ECS and BAS (Bleed Air System) are sub-systems which should be developed by partners. BAS provides air from engines to different direct consumers like ECS and WAIS (Wing Anti-Icing System).

Predefined scenarios are presented in the library Tests package, as shows in fig.24.

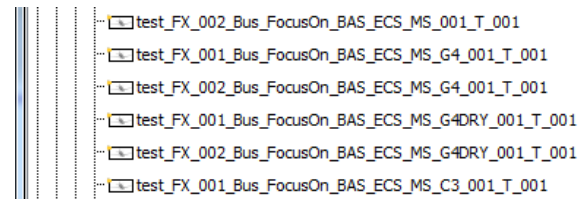


Figure 24. Set of models for architecture assessment

Such a predefined scenario is represented in the following figure. This part of the global architecture is limited to BAS and ECS that should be developed by partners (LTS ...) and to the boundary sub-systems, provided by the Aircraft Integrator as models.

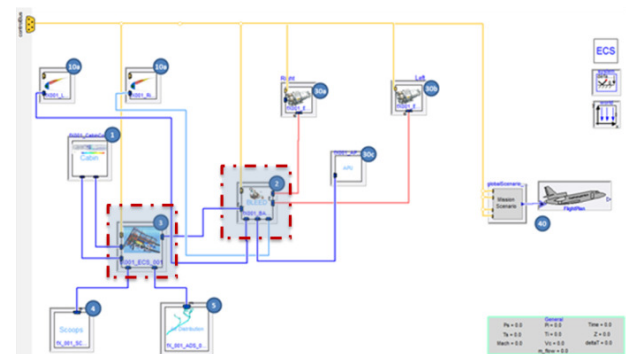


Figure 25 Architecture FX_001 focus on BAS and ECS

Elements of the model:

- (40): Scenario (Mission) and environment
- (2): Bleed Air System (BAS)
- (3): Environmental Control System (ECS)
- (30a et 30b): Propulsion System (Engines)
- (10a et 10b): Wing Anti-Icing System (WAIS)
- (1a et 1b): Cabin and Cockpit
- (4): Scoops
- (5): Air Distribution System
- (30c): Auxiliary Power Unit (APU)

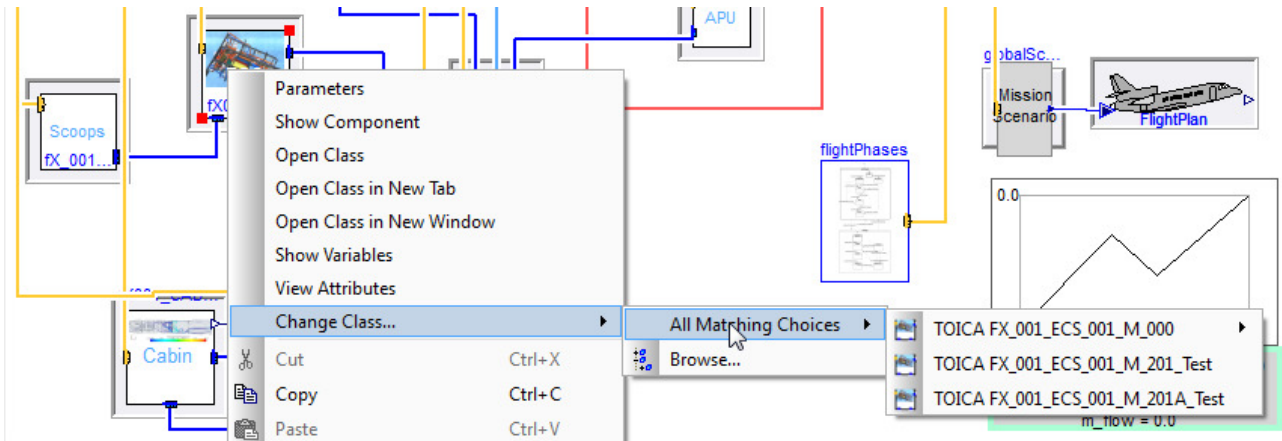


Figure 26. selection among replaceable models constrained by the predefined interface

The aim of the test cases is to allow the partner to check the behavior of their systems integrated with boundary sub-systems, and adjust them to comply with requirements. The different test cases must be delivered by the partners as part of evidence of the right technical choices made by them.

These test cases are also useful to guaranty that models developed by partners are able to simulate correctly with predefined boundary models, scenario and solver configurations. This may serve as acceptance tests of models at partner’s offices, which must give the same results in similar conditions at integrator facility, before final integration.

The models of partners will be used to study the integration of partner sub-systems within the global architecture with the interaction with more detailed models only available at Dassault Aviation, with other sub-systems, or to make trade-offs between different architectures.

- The required parameters and published variables,
- A clear designation. Each model must be clearly identified like defined in the following table to assure models versioning and traceability.

Table 1 Set of models involved within design.

Icon	Interfaces	Variants	Objective
	IF_FX_001_CAB_001	FX_001_CAB_001_M_001	Cabin simple model
		FX_001_CAB_001_M_002	Cabin detailed model
		FX_001_CAB_001_M_003	Cabin detailed model

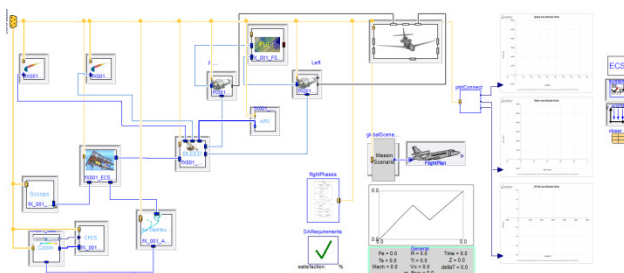


Figure 27. Partners models integrated within Dassault Aviation environment (flat view)

According to analysis needs, partners should provide models of sub-systems with different characteristics, in particular with:

- The integration and connection of their models to predefined interfaces (extended from Interfaces connectors package). See example fig.28.
- The right level of details to get the right behavior according to the specified analysis (stationary, dynamics ...)

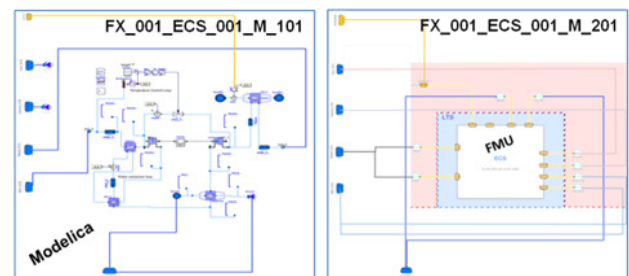


Figure 28 Example of models which inherit of the same interface

5.2.2 Sub-systems checks

It is required to provide models as Modelica models or FMU embedded within Modelica.

The purpose of using Modelica and FMU is to use standards. It is then important to be able to check that Modelica models and/or FMUs are in accordance with the specifications. For this, Modelica Association provides tools to check these compliances.

It is also requested to check each model individually, using at minimum test benches provided in the library, as represented in the following figure.

Several of these models are surrogate models built from 3D CFD or FEA simulations, which could be embedded either in Modelica or FMU like already done in project CSDL (E. Thomas *et al*, 2012).

5.3.4 Problem to tackle

The remaining problem for simulation of complex model is to get sufficiently quick simulations to be able to make assessments and take decision on time.

Complexity could come from heterogeneity of models, which leads to various dynamic phenomena coupled together. It could also come from Modelica power which allows strong coupling of dynamic physical model to control model with discrete or synchronous features. Finally, it could come from FMU allowing transformation of models as black boxes, which is not always compliant with efficient simulations.

But before simulation, it is required to be able to initialize the models. The convergence of Newton Solver during initialization is a key challenge. It is particularly important for fluid systems which often lead to solve complex non-linear and stiff equations, and numerical oscillations at the beginning of the simulation.

Several strategies have been developed, such as the homotopy method or with some decoupling especially during initial phase.

All these simulation issues tend to find solutions with improvement of Modelica tools and FMU definition. But, it will be also mandatory to be able to check easily that models are compliant with rules or requirements. For this current development within the project ITEA2 MODRIO, and initiated in EUROSYSLIB to observe models and check requirements will help designers.

6 From early design phase to detailed phase

6.1 Introduction

As soon as partners have been selected following RFP phase, Development phase will surge and architecture definition will continue to grow up to detail solution and bring them to reality. It is then mandatory to use strong process and associated tools to manage sub-systems and partners all together and in a consistent way, from top requirements to solutions as illustrated in fig.34. For these activities Dassault Aviation, like other Aircraft integrators, uses PLM environment (Product Lifecycle Management).

The following picture illustrates how can be managed information from an architecture level to another with traceability links from high level requirements to the DMU of the whole aircraft.

Within a previous project, CSDL, the links between requirements and simulation has been already analyzed successfully, but still with few models. The target of

TOICA is to develop the process and tools to enable handling of more complex systems, close to actual aircraft systems.

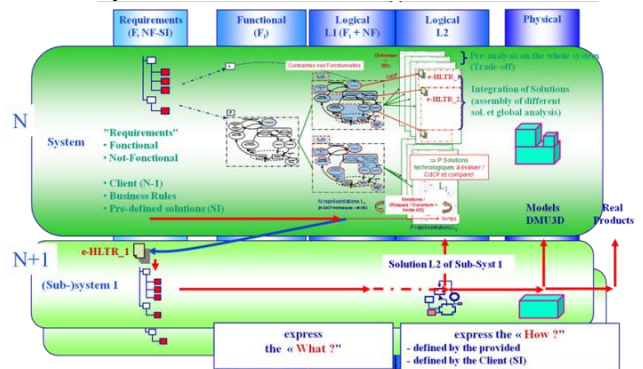


Figure 32. RFLP process and need of systems analyses

6.2 PLM Integration

In Toica, one of the investigated major topics is to build up a so called “Architect Cockpit” to help Architects (at Aircraft level, sub-systems levels for managing efficiently alternatives and trade-offs in a collaborative way). The target is in particular to build-up and demonstrate ability to manage in a collaborative context models with multi-levels of details for making trade-offs between architectures, with close operational design.

Dassault-Aviation uses for this purpose the 3D Experience platform from Dassault-Systèmes, which tightly integrates traditional a common 3D Digital Mock-Up (DMU) with system engineering activities, including simulation. In particular it allows managing roles of users, design workplace to work within defined teams or to share information with other partners.

The purpose of this paragraph is to show that the process and tools described before, dedicated to RFI/RFP phases, are fully compatible with the Development design phase, with a progressive and smooth integration.

The architectures hierarchy defined in fig.1 has been reproduced in Catia Systems, with two alternatives (a traditional alternative, and a More Electrical one). One of the alternatives is represented in the fig.34, with highlight of the Air sub-system.

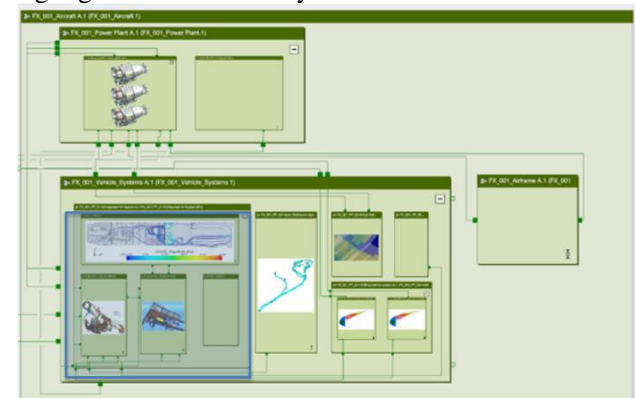


Figure 33. Representation of the traditional Architecture

As illustrated below, internal models and partner models can be attached to each sub-systems or equipments.

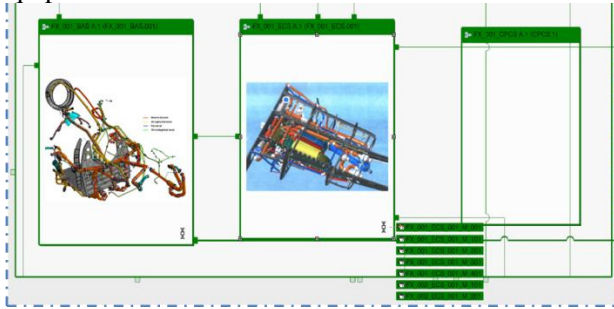


Figure 34. Replaceable models attached to ECS

Associated models, with their interfaces, are integrated within each system as represented below.

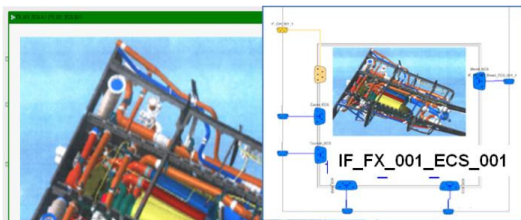


Figure 35. One of the model attached to ECS

Even quickly described above, it can be shown that the models for early phases and development phases are managed in the same repository:

- Dassault Aviation framework, as encrypted Modelica/Dymola libraries, provided to partners can be generated automatically
- Models provided by partners can be quickly added to the models attached to sub-systems in V6 3DExperience platform, and are automatically compatible if partners have integrated them according to required interfaces.

It is then possible to use V6 tools 3DExperience environment to define and manage scenarios for testing subsystems and making trade-offs, which can be recorded in V6 database, to be modified or replayed further as illustrated in (fig.36).

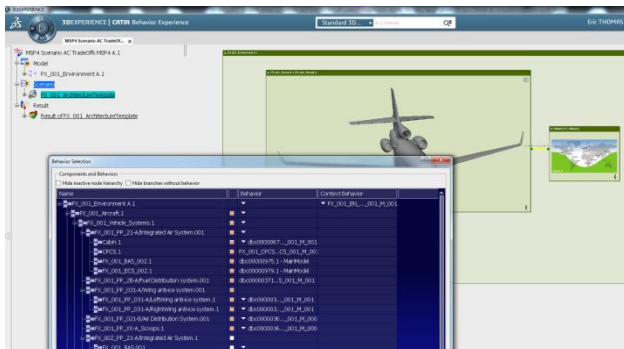


Figure 36. Scenario within V6 3D Experience

7 Conclusions

This paper has highlighted needs for aircraft systems design. It has also described the new process being developed within FP7 TOICA project. It brings out solutions to allow easier handling of complex systems assessments in a better and more flexible way.

In the current process, the aircraft architects ask suppliers to provide models for multi-systems assessments into Aircraft integrator office. This process has two drawbacks. It is difficult to assure that models will run in integrator's facility, and this request can't be currently handled efficiently during very early phases like RFI or RFP phases.

The new process, based on model exchanges using a Modelica framework, allow more efficiency and flexibility. As demonstrated, the workflow for architecture and sub-system assessment is straightforward compared to current one. In addition, it may more deeply imply partners in assessment success, and may allow finding more innovative solutions by opening up aircraft requirements.

The process and associated tools are based on current powerful capabilities of Modelica and FMI which continue to improve to be able to manage heterogeneous models required for Aircraft systems assessment.

This paper has also briefly described some elements of the further phases to show that the new process is fully compatible, and takes advantage of tools associated, to development phases.

Acknowledgements

This work was partially supported by the French government through the FP7 TOICA (Thermal Overall Integrated Conception of Aircraft) and ITEA2 MODRIO (Model Driven Physical Systems Operation) projects.

References

- Martin Malmheden, Jean-Baptiste Quincy, Michel Ravachol, Eric Thomas. "CSDL - Collaborative complex system design applied to an aircraft system". *Modelica Conference*, No 9, pp. 855–865, 2012. DOI: 10.3384/ecp12076855
- Mike Dempsey, Magnus Gäfvert, Peter Harman, Christian Kral, Martin Otter, Peter Treffinger. "Coordinated automotive libraries for vehicle system modeling". *Modelica Conference*, No 5, pp. 33–41, 2006.
- Gertjan Looye. "The New DLR Flight Dynamics Library". *Modelica Conference*, No 6, pp. 193–2012, 2008
- Philip Jordan, Gerhard Schmitz. "A Modelica Library for Scalable Modeling of Aircraft Environmental Control Systems". *Modelica Conference*, No 10, pp. 599–608, 2014. DOI: 10.3384/ECP14096599
- Bettina Oehler. "Modeling and Simulation of Global Thermal and Fluid Effects in an Aircraft Fuselage". *Modelica Conference*, No 4, pp. 497–506, 2005.

Thorben Vahlenkamp, Stefan Wischhusen “FluidDissipation for Applications - A Library for Modelling of Heat Transfer and Pressure Loss in Energy Systems”. *Modelica Conference*, No 7, pp. 132-141, 2009. DOI: 10.3384/ecp09430012

Francesco Casella, Hilding Elmqvist, Rüdiger Franke, Sven Erik Mattson, Hans Olsson, Martin Otter, Michael Sielemann “Stream Connectors – An Extension of Modelica for Device-Oriented Modeling of Convective Transport Phenomena”. *Modelica Conference*, No 7, pp. 108-121, 2009. DOI: 10.3384/ecp09430078

Francesco Casella, Rüdiger Franke, Katrin Proelss, Martin Otter, Michael Sielemann, Michael Wetter “Standardization of Thermo-Fluid Modeling in Modelica.Fluid”. *Modelica Conference*, No 7, pp. 108-121, 2009. DOI: 10.3384/ecp09430077

Michael Wetter “Modelica Library for Building Heating, Ventilation and Air-Conditioning Systems”. *Modelica Conference*, No 7, pp. 393-402, 2009. DOI: 10.3384/ecp09430042

Daniel Bouskela, Laurent Chastanet, Audrey Jardin, Sandrine Loembé, Thuy Nguyen, Nancy Ruel, Raphaël Schoenig, Eric Thomas “Modelling of System Properties in a Modelica Framework”. *Modelica Conference*, No 8, pp. 497-506, 2011. DOI: 10.3384/ecp11063579

Martin Otter, Nguyen Thuy, Daniel Bouskela, Lena Buffoni, Hilding Elmqvist, Peter Fritzon, Alfredo Garro, Audrey Jardin, Hans Olsson, Maxime Payelleville, Wladimir Schamai, Eric Thomas and Andrea Tundis “Formal Requirements Modeling for Simulation-Based Verification”. *Modelica Conference*, No 11, 2015.

Websites

- Modelica Association website: www.modelica.org
- FMI website: <https://fmi-standard.org>
- ITEA2 EUROSYSLIB project: www.eurosyslib.org
- ITEA2 MODELISAR project: www.modelisar.com
- FP7 TOICA project: <http://www.toica-fp7.eu/>
- ITEA2 MODRIO project: www.ITEA2.org/ Modrio
- OMG SysML: www.omg.sysml.org

List of Acronyms / Abbreviations

Acronym / Abbreviation	Definition
A/C	Aircraft
APU	Auxiliary Power Unit
BAS	Bleed Air System
CAB	Cabin
CAU	Cold Air Unit
CPCS	Cabin Pressurization Control System
DASSAV	Dassault Aviation abbreviation within TOICA
DMU	3D Digital Mock-Up
ECS	Environmental Control System
FHA	Functional Hazard Analysis
FMI / FMU	Functional Mock-Up Interface / Unit
ICD	Interface Control Document
IAMS	Integrated Air Management System
IP	Internal Property

Acronym / Abbreviation	Definition
LTS	Liebherr Aerospace Toulouse
MSL	Modelica Standard Library
Pax	Passengers
PLM	Product Lifecycle Management
RFI / RFP	Request For Information / Proposal
WAIS	Wing Anti-Ice System

Annexe

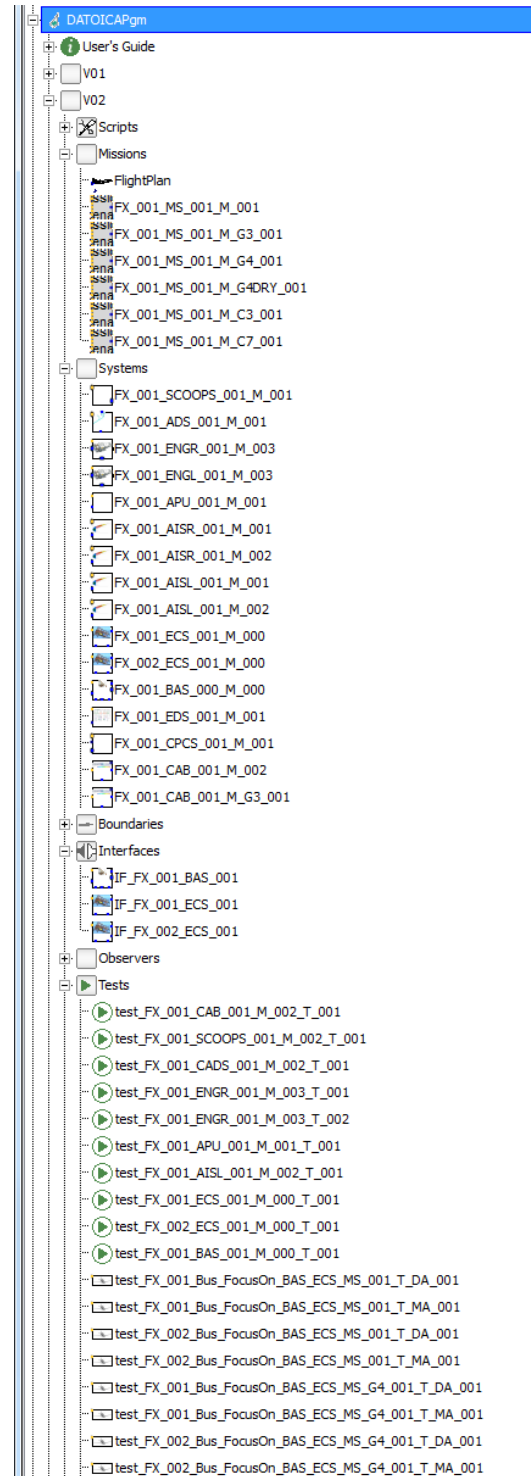


Figure 37. Library content

Using FMI in a cloud-based Web Application for System Simulation

Stefan Bittner Olaf Oelsner Thomas Neidhold

ITI GmbH, Germany, info@iti.de

Abstract

This paper presents a generic approach to combine cloud computing and system simulation. It shows the benefits of using FMI to deploy self-executing simulation units on multiple machines. Besides managing the calculation itself, we also present a web interface for uploading, managing and analyzing simulation models. To benefit from available hardware resources in the cloud, an engine is integrated which allows the definition of multiple simulations with different parameter sets in a single step.

Keywords: FMI, Cloud, Web, System Simulation, Parameter Study

1 Introduction

Simulation calculations can be very demanding for hardware resources, such as computing power and disc space. If these resources are not needed permanently, there may be a conflict of insufficient resources to finish a project within a certain time on the one hand, and paying too much for resources on the other hand. Cloud computing could be a solution here: cloud providers offer a wide range of resources which can be allocated on demand and used from anywhere around the globe.

However, reserving raw resources, such as a plain virtual machine, and setting the simulation environment up that is needed for a calculation can be a time consuming task. Let alone additional licenses for the simulation tools which would be necessary in order to run them on the remote machine. Although easy to setup, remote resources in the cloud are not a trivial task to deal with, especially not for simulation engineers. Offering simulation services over the web, which make use of available cloud infrastructure, seems to be a much better approach. The advantages and possibilities of such a service have been presented in (Tiller, 2014). We are extending this idea to a more generic approach: a web service based on FMI itself instead of a specific simulation model where the users are able to upload and share their own models and results.

2 System Perspective

We have implemented different approaches in two projects we are involved in, Cloud4e (Cloud4e, 2012), funded by the Federal Ministry of Economics and Technology, and CloudFlow (CloudFlow, 2013), funded under the 7th Framework Programme of the European Commission. While in Cloud4e, a REST service has been developed which communicates via the Open Cloud Computing Interface (OCCI), in CloudFlow a SOAP service is embedded in another service architecture as part of workflows. Instead of explaining these implementations in detail, which lies outside the scope of this paper, we want to present a brief overview of the underlying service architecture and how it works. For a more detailed description, see (Limmer et al., 2014) and (CloudFlow, 2013).

2.1 Architecture

The web service consists of different services, each of which is necessary to perform a specific task. A storage service stores models and results and protects them against unauthorized access. The calculations are run by a calculation service which has access to the storage in order to obtain the model, read its configuration and write back result data. Everything is monitored by the simulation web service. It manages models and data, authorization and cloud resources. This service is the heart of the architecture organizing everything that is necessary for this service to work. Different application types, such as web applications as well as desktop applications, can communicate with the simulation web service to request data, upload models or submit simulation tasks to be calculated by the underlying infrastructure.

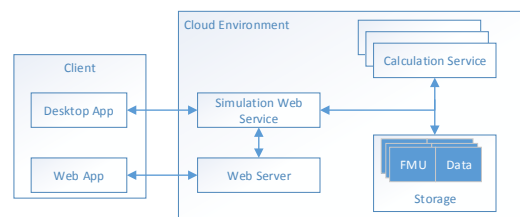


Figure 1. Service architecture

Every component in the architecture may reside on a different logical computer, but does not have to. For example, the web server can be hosted on the same machine as the simulation web service, while calculations can be run on separate machines simultaneously or only on one machine, one at a time. The web server shown in figure 1 provides a web interface, which serves as the front end for the user, and is discussed in detail later on.

The future architecture might include other services as well, for example external authentication and storage services.

3 User Perspective

The user does not have to care about the underlying system, with one exception: since allocated resources cost money, it is desirable to know and control how much resources are in use or will be in use. This is only one example that shows that a stable communication between system and user has to be established. This is realized through a web application which is connected to the service and provides a web platform based on HTML5 and JavaScript. As such, it is accessible through any modern browser from any device connected to the internet. It provides the user view and control of the simulation service: its content and its state.

Models, Tasks, Simulations When an FMU is uploaded, it is embedded in a model where it serves as the description of the simulation model. This description can then be used to create multiple instances of the model called "tasks". A task contains a description for each parameter representing a configuration of the model. They belong to one model at any time.

For the system simulation service, a task is an executable unit: it references an existing FMU and contains a description of its parameters. It is used as input for the calculation service. The results are assigned to the task as simulations which in turn contain the corresponding parameter and result set. Why is there a parameter configuration in the task and a parameter set in the simulation? They may differ, and a task may contain multiple simulations as described later on.

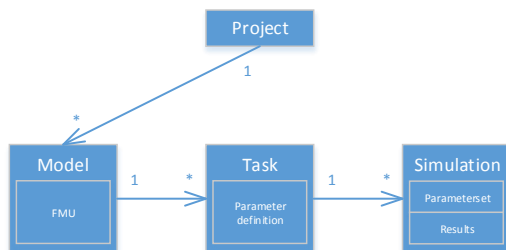


Figure 2. The FMU is embedded in a simple structure to support access management and multiple simulation results.

Projects and Users Models can be assigned to projects in order to organize and share them. While this mechanism can be used for collaborative work and project management, it is also possible to give other users restricted access to models, for example, to the simulation results without rights to modify the model. Possible restrictions are shown in table 1.

Project	Model
<i>General</i>	
Create	Upload
Delete	Open
	Delete
<i>Contents</i>	
Add models	Modify tasks
Remove models	Execute tasks
<i>Right management</i>	
Add/Remove users	Modify access rights

Table 1. Access rights for projects and models which can be granted to users and user groups

We think that implementing a reliable project and user management adds a significant value to simulations in the cloud. Sharing models and presenting results are true benefits besides the use of external resources for the calculations themselves. However, user and access management is still under development.

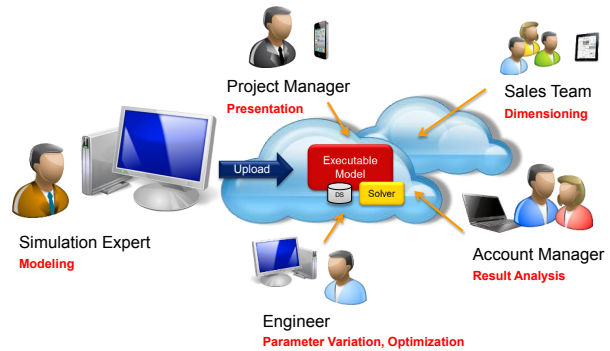


Figure 3. Scenarios for sharing simulation models in the cloud.

Configuring multiple Simulations The parameter configuration of a task may contain range expressions for one or more parameters. A range expression defines more than one value for this parameter. There are currently two options available: comma separated list (<value>,<value>) and sequence constructor (<startValue>:<stepSize>:<endValue>). When multiple parameter values are defined, the simulation service creates a parameter set for each combination, which

results in a vast number of simulations. Each of them is calculated and, after that, part of the result of the corresponding task.

volumeFlow.interval	200,400	ms
thermalConductor.G	1:1:10	Constant thermal conductance
heatCapacitor.C	0.1:0.1:1	J/K

Figure 4. Defining multiple parameters values with the result of 200 different parameter sets.

3.1 Web Interface

The web interface, a .Net MVC web application, gives access to the contents of the simulation service through any modern web browser. After logging in, the user has access to the contents and functions he is authorized to use. The SignalR library allows for a bidirectional communication between client and server, providing a user experience similar to desktop applications. As the main interface between user and simulation service, the web application allows the user to manage models and tasks, request calculations and results.

Model Management The model list shows projects and models and allows the user to create new projects, move models from one project to another and upload an FMU from the local disk to create a new model. A preview is provided for each model showing the embedded image of the model and information, such as author, date of creation and number of tasks. Each model can be deleted and opened here.

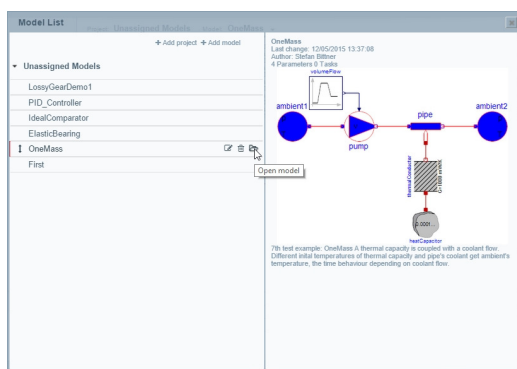


Figure 5. Web interface: List of models and model preview.

Task Management Once opened, the model contents are displayed in the web application. The taskbar allows the user to create and delete tasks as well as start, stop and reset the calculation state of each task. Each task contains a parameter configuration which can be modified as long as the task has not been executed. If available, the model structure is displayed in the center. Model elements can be selected to filter the list of parameters.

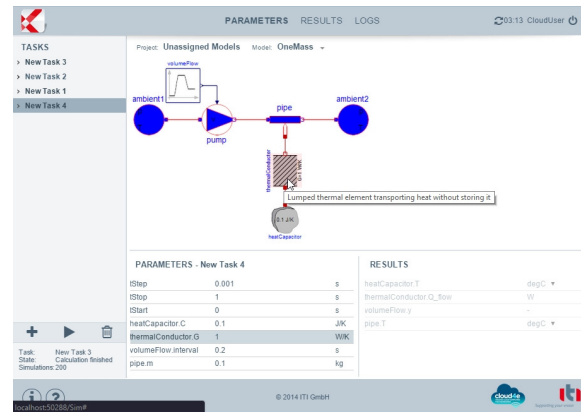


Figure 6. Web interface: Task list and parameters, interactive image of the model in the center.

Result Analysis The result panel contains a graph where the transient results of one or more output parameters are displayed. This may be a set of curves per output parameter when multiple tasks are selected for result analysis, while each of them may contain multiple simulations. Such a set of results also contains a set of parameter configurations, one for each simulation. The range of varying parameters can be modified using a slider control which changes the number of curves visible in the result graph. A CSV file download is available for further analysis.



Figure 7. Web interface: result view.

4 The Role of FMI

In this scenario, two sides had to be addressed: creating a scalable service structure capable of running in a cloud environment, and creating a web application to access simulation parameters and results. FMI works either way: while the XML model description can be used to create a generic interface to access model contents, the integrated solver can be embedded in web services and deployed on any machine. Furthermore, there is no restriction to the contents and size of the model as long

as it meets the FMI specification. No additional dependencies or libraries are needed: everything is packed into the FMU which, of course, can be downloaded and integrated in other toolchains as well.

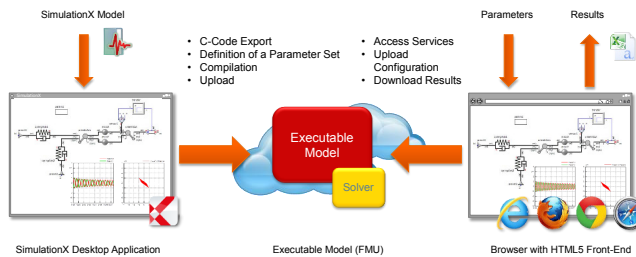


Figure 8. FMI in the cloud: scenario using SimulationX

FMI in the Cloud To run an FMU on a machine in the cloud, a web service had to be implemented which is able to consume the C interface, run as a master for the functional mockup unit and communicate over the web. Since the simulation solver is embedded in the FMU, this service is only an adapter between the FMU and other web services.

Model Description and User Interface The model description itself can be used to create a generic user interface to visualize and gain access to parameters and result variables exposed by the FMU. Different units can be used to display/define parameter values if these units are available in the model description. Optional contents can be added to the FMU's resources folder to be used in the web interface. For example, SimulationX adds an image and interaction map of the model.

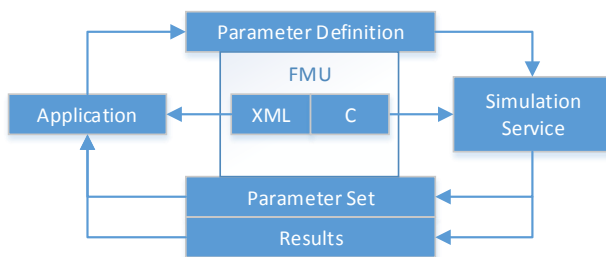


Figure 9. FMU between application and simulation service

5 Conclusion

This paper presents a web application using a cloud service architecture to manage, store and perform simulation calculations and their results. FMI 1.0 for Co-Simulation is the key technology for the simulation part of the service. It offers an easy-to-use scenario to deploy calculations on remote machines and contains a description to generate a generic web interface. This allows us to

support a wide range of simulations, especially because creating FMUs using the FMI specification is supported by a growing community of simulation tool vendors.

There is currently still some work to be done, especially regarding user access management. For the future, this service can be extended, for example, to combine FMUs for online co-simulations, or integrating other services, such as PLM services, where FMUs are integrated together with other information to represent a product lifecycle.

References

- Cloud4e. Trusted Cloud Computing for Engineering, 2012. URL <http://www.cloud4e.de/>. Funded by the Federal Ministry of Economics and Technology (BMW).
 CloudFlow. Computational Cloud Services and Workflows for Agile Engineering, 2013. URL <http://www.eu-cloudflow.eu/>. Funded under the 7th Framework Programme of the European Commission.
 S. Limmer, A. Ditter, M. Srba, S. Thomas, A. Schneider, S. Rülke, O. Oelsner, A. Uhlig, S. Schmitz, D. Fey, and C. Boehme. The Project Cloud4E – Cloud Solutions for Engineers. *Lecture Notes in Computer Science*, 2014.
 T. Neidhold, S. Bittner, and O. Oelsner. SimulationX Goes Online – A Web Platform for Cloud-Based Simulation. In *ITI Symposium*. ITI GmbH, 2014.
 T. Neidhold, O. Oelsner, S. Bittner, A. Ditter, and D. Frey. Rechnen in der Wolke. *Digital Engineering*, 2015.
 M. Tiller. Vehicle Thermal Management – A Case Study in Web-Based Engineering Analysis. *Proceedings of the 10th International Modelica Conference*, 2014.

Anticipatory Shifting – Optimization of a Transmission Control Unit for an Automatic Transmission through Advanced Driver Assistance Systems

Salim Chaker¹ Michael Folie² Christian Kehrer¹ Frank Huber²

¹ ITI GmbH Dresden, Germany, {chaker, kehrer}@ititim.com

² IPG Automotive GmbH, München, Germany {michael.folie, frank.huber}@ipg.de

Abstract

By integrating system simulation with vehicle dynamics into real-time environments, it is possible to simulate the physically correct behavior of vehicle components and also adjust it to the required operating strategies depending on external factors. Multi-physics system simulation for realistic representations of powertrains and their behavior combined with dynamic driving simulation allows for optimizations of the transmission control unit for an automatic transmission by employing advanced driver assistance systems for an increased efficiency through anticipatory shifting.

Realistic load cases that are based on measured data help optimize fuel consumption and driveline dynamics with respect to the control algorithms by using variation calculations with variable transmission parameters. This works also the other way around when control algorithms are validated and optimized quickly and free of risk as part of rapid prototyping.

Keywords: Automatic Transmission, Advanced Driver Assistance Systems, CarMaker, SimulationX, Modelica, FMI

1 Introduction

During the development of modern transmissions in the automotive sector – as for any technical system – there is a growing complexity caused by more and more features and functionalities. In order to get reliable test results for making a profound decision, it is inevitable to integrate such interacting functionalities into a virtual prototype, which has already become common practice. Especially advanced driver assistance systems monitoring the surrounding area through a number of sensors to generate an environmental model for situation-based interpretations contribute to the growing relevance of simulation solutions.

The more and more diverse interactions between the many sub-systems spanning across various physical domains, such as mechanics, electronics, hydraulics or control engineering, can be modeled in an object-oriented way including their dynamic effects. With a higher efficiency in mind, these multi-physics models are used for various scenarios during the development and analysis of new transmissions, such as hybridization, cylinder deactivation or the integration

of advanced driver assistance systems. The optimal utilization of available potentials can only be achieved by combining means of transmission tuning and optimizing operating strategies. As a consequence, advanced driver assistance systems are more and more developed with the focus not only on safety aspects, but also on the reduction of fuel consumption and on anticipatory driving. Many functionalities and applications are already in use or are being developed. Tools which can perform certain tasks of engineers in a virtual world including software development, applications and testing are becoming increasingly indispensable. The objective of the work presented in this paper is to develop a virtual verification environment for gearbox control algorithm with focus on fuel consumption.

2 Modeling an 8-speed automatic transmission

2.1 Modeling paradigm

For the representation of technical systems, there are a number of modeling options available. A principle categorization may be based on the interactions within a model. Signal-based modeling (e.g. Matlab/Simulink, ETAS ASCET) shows the functional correlations based on the principles of control engineering by using blocks with inputs and outputs. This includes the direction of effect and thus the solution direction, i.e. causal modeling. As real physical systems barely show a direction of effect, it is often necessary to solve the differential equation systems that describe the technical system. Especially complex systems with multiple components which are connected on a differential-algebraic level may push single-oriented models towards their limits.

Acausal approaches are not subject to such limitations so that also huge systems can be modeled intuitively. Instead of inputs and outputs, there are connectors functioning as interfaces for flux variables (e.g. current, inertia, force) and potential variables (e.g. voltage, angle, distance). The overall equation system is a result from the inner system equation of the components and the equations from the connections (potential variable equation, flux variable balance). This declaratively described equation system is symbolically transformed and solved in advance

allowing for a very time-efficient simulation. The most prominent example for object-oriented, acausal modeling is the Modelica language (P. Fritzson, 2004).

The major features of both signal-oriented and object-oriented acausal modeling are listed below (tab. 1) for the respective representative Matlab/Simulink and Modelica/SimulationX (ITI GmbH 2013). For

modeling the transmission's control loop, object-oriented modeling is the preferred option, while signal-oriented modeling in Matlab/Simulink is the tool of choice for vehicle parts related to control engineering (engine and transmission control units, operating strategies).

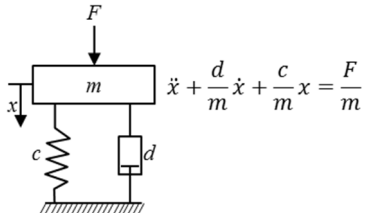
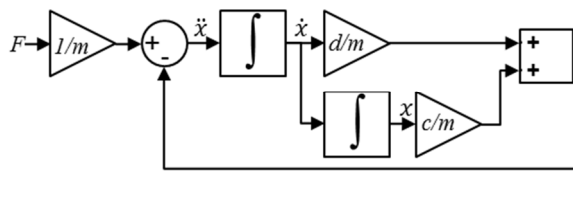
Acausal, object-oriented modeling (Modelica/SimulationX)	Causal, signal-oriented modeling (Matlab/Simulink)
	
<ul style="list-style-type: none"> • Direct implementation of differential-algebraic equations to describe technical systems • Free solution direction (acausal modeling with differential-algebraic equations) • Symbolic analysis and forward solution of the equation systems (order reduction), then numeric integration (→ minimizing the computational effort) • Easy creation of complex systems by connecting them through interfaces similar to real systems • Models with identical structure to real systems (high model readability) • Class structure and inheritance (high re-usability, extendibility) 	<ul style="list-style-type: none"> • The implementation of differential-algebraic equations requires reformulating/rearranging them in order to define the solution direction (or re-modeling an inverse model if needs be) • Numeric solution block by block, no analytic summary or forward solution • Time-consuming creation of complex systems through manual solution of the involved differential equation systems • High level of abstraction of the real physical system (bad model readability) • No inheritance or class structure (bad re-usability and extendibility)

Table 1: Comparison of modeling paradigms

2.2 Model of the 8HP transmission

The ZF 8HP is an 8-speed automatic transmission by ZF Friedrichshafen AG. The transmission consists of four planetary gear sets and five shift elements (three multiple plate clutches and two brakes). Figure 1 shows the interaction between the many different elements involved. Gears are changed by opening and closing the shift elements. Depending on the selected gear, the shift elements are open or closed in order to achieve a certain transmission ratio by adapting the distribution of force through the planetary gears. This is handled by a transmission control unit which was modeled in Matlab/Simulink to take advantage of the benefits of signal-oriented modeling.

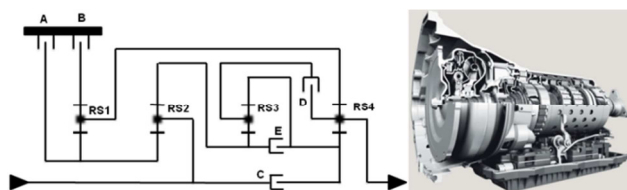


Figure 1. Transmission diagram of the ZF 8HP automatic transmission

SimulationX is a continuous CAE solution for acausal/object-oriented modeling, simulation and analysis of physical effects. The tool comprises ready-to-use model libraries for 1D and 2D mechanics, 3D multibody systems, power transmission, hydraulics, pneumatics, thermodynamics, electronics, electric drives, magnetics and control engineering. The 8HP transmission including its hydraulic actuator structure was modeled in SimulationX with elements from the 1D rotary mechanics library and with signal blocks (see Fig. 2). The model received also an interface for inputs and outputs matching those of the gearbox modeled on the open integration and test platform CarMaker in order to allow for a seamless integration.

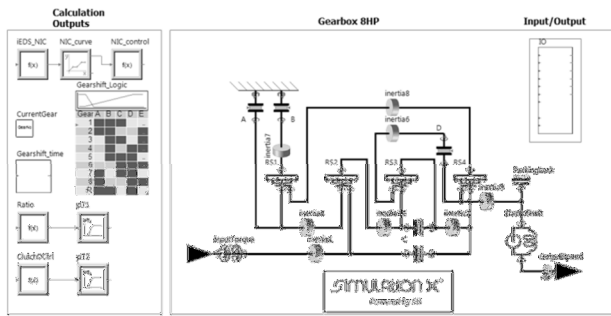


Figure 2. SimulationX model of ZF's 8HP automatic transmission.

The desired gear or sequence of gears for dynamic gear changes can be assigned to the mechatronics module `schaltLogikVerz` (see left side of Figure 2) that implements the controls of the brakes (A, B) and the clutches (C, D, E) of the 8HP transmission. Clutch and brakes were modeled with elastic friction points including signals for (de-)activating the torsional flux between the drive components through a controller.

2.3 Model exchange via FMI

Due to the high interoperability between SimulationX and CarMaker, it was an easy choice to go for a dual approach combining the benefits of both tools. In order to implement such a simulation concept, model exchange capabilities or a coupled simulation between the chosen tools is inevitable. This has the following advantages:

- Platform independent availability of component models for OEMs
- Black box models for IP protection during model exchange between companies
- Option to use models from different modeling environments with certain characteristics and benefits that can improve the model quality and simulation efficiency
- Re-usability of models regardless of the modeling environment (FMI (Functional Mock-up Interface) standard [3]) as well as easy testing and validation of models
- Realization of distributed simulations for coupled models from different domains and for the optimal utilization of the available resources

This principle was further promoted with the FMI standard specified in the MODELISAR project [3]. This interface standard allows for transfers functionalities of an entire SimulationX model in a FMU (Functional Mock-up Unit). Other simulation environments (e.g. CarMaker) can then instantiate the generated FMU and can access the unit's functionalities through the FMI interface (FMI Association, 2010). There are two types of the FMI

standard: FMI for model exchange and FMI for co-simulation (see Figure 3).

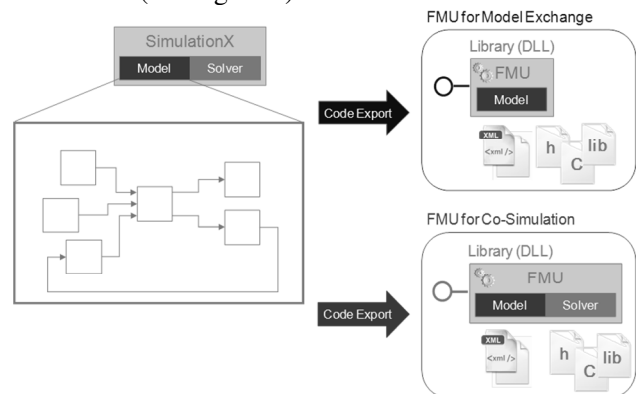


Figure 3. FMI code export for model exchange and for co-simulation.

While for FMI for model exchange, the model's behavior is exported as C code and entirely integrated and simulated in the target simulation environment, FMI for co-simulation involves two or more simulation tools that are coupled with each other. The model's behavior is exported as C code together with the solver. The generated FMU for co-simulation runs as a slave in the target simulation environment and exchanges the computed results at discrete communication points with the master environment. Between these points, the individual solvers compute the sub-systems independently from each other.

3 Integration and test platform CarMaker

CarMaker and the related products TruckMaker and MotorcycleMaker offer plausible models for driving physics and driving environment as well as virtual drivers capable of performing relevant maneuvers of ordinary drivers, but also of test drivers. As a test platform, CarMaker provides also maneuver characterizations based on real driving behavior. Also complex open and closed loop test scenarios can be realized.

Besides vehicle and driver models, CarMaker also includes a complete environment simulation with streets (curves, traffic signs, traffic lights etc.), moving traffic with near field sensors and the integration with digital maps (e.g. ADAS-RP by HERE, Google Earth). This creates a very realistic representation of the desired test environment. CarMaker also allows for a consistent implementation of the X-in-the-loop approach (XiL) (Figure 4). The XiL method permits early integration into the complete vehicle and comprehensive validation of relevant system components, be it as model, software or hardware (B. Schick, 2013).

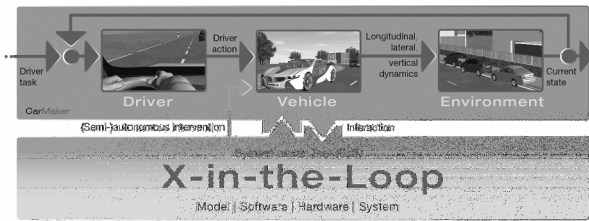


Figure 4. XiL permits early verification of systems. CarMaker provides the necessary interfaces to integrate all relevant components and systems into the virtual vehicle.

As an open integration and test platform, CarMaker offers an interface architecture tailored to the needs of the automotive industry (Figure 5). At the click of a button, models, software components and actual vehicle components are integrated into so-called digital prototypes – from the single component to integrated systems. If necessary, individual powertrain components, the chassis, assistance or control systems, for instance, but also concepts of operations or display functions can be integrated (B. Schick, 2012). The virtual integration is the basis for the analysis of occurring influences of the tested components and functions on the vehicle’s behavior. Flaws in the development process can thus be identified early on (S.-A. Schneider, 2012).

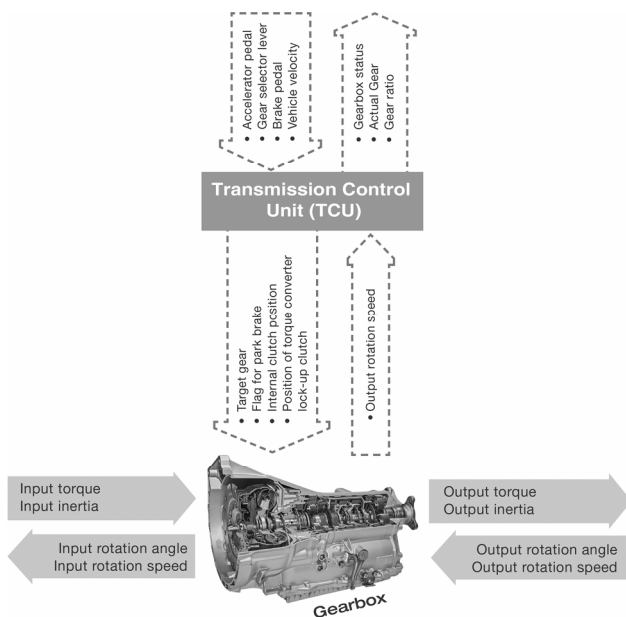


Figure 5. Realistic interfaces with gearbox components are necessary for an easy and continuous integration. Additional inputs and outputs can be included.

The digital prototype systems can be verified as a whole during virtual test driving. Such test drives can be repeated and modified if needed, while the test results are valid up to the physical limits. Virtual test driving and realistic driving scenarios permit evaluations of new developments across the entire V model (H. Palm, 2013).

4 Driver assistance systems in the powertrain and the gearbox

Today’s automatic transmissions must meet various requirements, such as weight, design space and shift comfort, but also aspects for reduced fuel consumption. As mentioned earlier, assistance systems can register and interpret environmental information and pass it on as data to systems and components.

The following section focuses primarily on shift strategies for which the mechanics module schaltLogikVerz received additional software modules. It demonstrates how these functionalities can be ensured and optimized in virtual test drive scenarios. This involved the transmission model needs to be integrated as a SimulationX FMU with the controller logic for the gearbox as a Simulink FMU in CarMaker. The software for the shift strategies has access to additional environmental information (Figure 6):

- Curvature of street curves
- Slope of the street
- Distance to the vehicle in front

CarMaker obtains the information about curvature and slope from the navigation system via the standardized ADASIS protocol. The distance to the vehicle in front is determined through a radar or camera model. All data captured through sensors are passed on to the transmission’s software and can be processed.

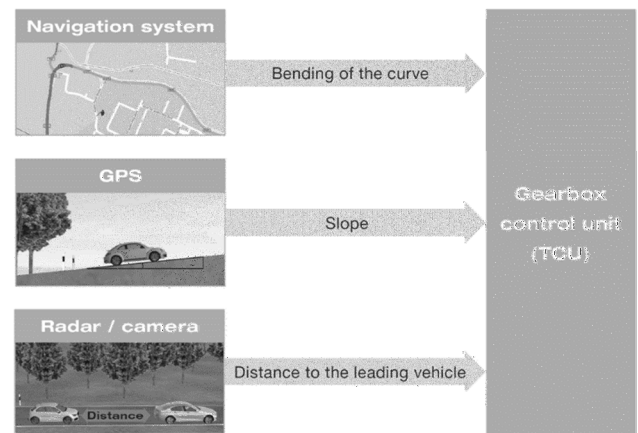


Figure 6. Information captured through sensors and passed on to the transmission’s software.

The aforementioned information can now be used for the development of such features as:

- Avoiding gear changes in a curve or finishing a gear change before the next curve,
- Engaging idle mode downhill to benefit from coasting,
- Using recuperation during deceleration or downhill to keep distance to the vehicle in front or to charge the battery.

The last two requirements are complementary. On the one hand, the kinetic energy should be used more effectively and the friction effects in the transmission should be reduced to a minimum, while on the other hand the battery should be charged through the wheel movements. A side-effect of this is that the engine slows down the vehicle which makes using additional external braking redundant at times, e.g. to keep the distance to the vehicle in front. Variation calculation can now help find the best gear change strategies for various slopes and distances to the vehicle in front. The integration of the transmission's plant model and software can be combined into various MiL, SiL, HiL, especially test bench, scenarios:

- MiL involves the transmission model and the software to be integrated in CarMaker as an FMU.
- SiL replaces the transmission's software with a production code which can be integrated in CarMaker as C code or as an AUTOSAR FMU.
- In the next step, the transmission software is replaced with a real control unit as part of a HiL setup.
- At the end, also the gearbox FMU is replaced with a real gearbox on a test bench.

CarMaker remains the integral tool throughout all development stages. The test cases created in the MiL phase as well as the vehicle model and the parameterization can now be used in all of the shown phases. The generated data reflects a high degree of reproducibility and is also suitable for testing and improving the transmission's software. The transmission test bench allows for certain physical effects to be included in the transmission model, which in turn contributes to a higher quality of the transmission control software.

It is then possible to also include the driver in further analyses. He can choose from either manual mode for overtaking maneuvers, for example, or switch to cruise or eco mode. These options can also be combined in several variations of different tracks and distances to the vehicle in front. That results in a large number of tests which can only be conducted in a reasonable fashion through MiL or SiL scenarios due to performance aspects. HiL and the test bench are only employed to test load cases for current, pressure and momentum, but also for certain limits of the defined spaces in order to minimize the number of tests.

Finally, driver input as well as vehicle and transmission data can be captured during the real test drive. The driver input and the velocity are used as input signals for the simulation in all phases in order to be able to compare the results from the various simulations with the actual test drive. Only a profound

model calibration can make a solid basis for variation calculations that are true to life.

5 Summary and outlook

The combination of shorter development cycles, an ever-growing model complexity, the level of detail and variation as well as the vast amount of reliable analyses to be conducted early on in the development process poses a huge challenge to an engineer's daily routine nowadays. Modeling, simulating and processing input data as well as the representation of the generated results in a digestible way require much effort. The object-oriented, acausal modeling approach meets the needs of the user, and the intelligent optimization algorithm delivers an efficient simulation. Due to the advantage of this modeling paradigm, more and more heterogeneous domains are combined with each other and simulated in just one model. The simulation performance with heterogeneous models reaches its technical limits. The implementation of a distributed simulation, however, through co-simulation as described above, for example, offer an excellent opportunity to master such challenges.

The separate integration of transmission and software offers several benefits – e.g. the reduction of the number of pre-defined variants or a quick way of virtually testing new concepts. Comparing real and simulated driving behavior creates a valid basis for virtual test driving. As a consequence, it is now not only possible to test and optimize safety relevant functions early on and efficiently in a closed toolchain, but also aspects of driving comfort and fuel consumption.

The described concepts also allow for fuel saving display functions in vehicles with manual transmissions. Such could include for example „Foot off the gas“ or „Shift to idle mode“. Other fuel saving features could be integrated with driver assistance sensors. Traffic sign and traffic light detection could be used to increase efficiency for coasting and recuperation phases. These ideas will be part of future developments. They will also focus on other aspects of virtual testing and the optimization of operating strategies for hybrid and non-hybrid vehicles, such as Car2X or car2car. After all, it is those test drives that are difficult or even impossible to conduct in real life that are the driving factor in the field of virtual validation.

References

- P. Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica No 2.1, 2004.
- ITI GmbH. SimulationX User Manual, Dresden 2013.
- FMI Association. Functional Mock-up Interface, MODELISAR (070006), 2010.
- B. Schick. Mission V-Process Enhancement by Integrated Vehicle Performance Evaluation within an Entire X-in-the-Loop Process, *Keynote SIAT ARAI*, 2013.

- B. Schick, V. Leonhard u. S. Lange. Vorausschauendes Energiemanagement im virtuellen Fahrversuch. *ATZ* No 4: 328 – 333, 2012.
- S.-A. Schneider, B. Schick, H. Palm. Virtualization, Integration, and Simulation in the Context of Vehicle Systems Engineering, *embedded world 2012*, 2012.
- H. Palm, J. Holzmann, A.-S. Schneider, H.-M. Kogeler. Die Zukunft im Fahrzeugentwurf, Systems-Engineering-Basierte Optimierung, *ATZ* No 6: 512– 517, 2013.

Simulation of distributed energy storage in the residential sector and potential integration of gas based renewable energy technologies using Modelica

Praseeth Prabhakaran Wolfgang Koeppel Frank Graf

German Technical and Scientific Association of Gas and Water (DVGW) Research station, Engler-Bunte Institut, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, prabhakaran@dvgw-ebi.de

Abstract

In-order to analyse the distributed supply and storage of energy in decentralised clusters, Modelica has been used to model buildings with micro Combined Heat and Power (μ -CHP) systems as their primary heat energy source. The classification of the buildings involve generalising their size based attributes. Therefore, different buildings, appropriate μ -CHP systems used inside them, the components for heat and electrical energy storage as well as associated control systems are modelled. The output power of μ -CHP systems and the dimensions of the storage units are chosen corresponding to the building size to account for space heating, warm water demand and electrical energy storage requirements. The control strategy used is heat prioritised where the power generated is either used in-house or fed back into the grid. Following the modelling of components, decentralised storage potential is analysed using distributed Power-to-Heat (PtH) as a storage strategy. To store the electrical energy locally, battery models are integrated with a power interface system. As an initial part of analysing distributed storage potential, various house types with μ -CHP units are simulated with measured weather dependent boundary conditions. Subsequently, potential integration of distributed storage into a larger storage strategy involving the electrical grid and the gas grid is discussed where the μ -CHP units could act as an interface enabling a symbiotic relationship between the power grid and the gas grid. *Keywords: micro CHP, Energy storage, Power to Heat, Building simulation*

1 Introduction

Studies regarding the implementation of smart energy systems based on decentralised production and consumption of energy show that new technologies like Micro Combined Heat and Power (μ -CHP) hold great potential. From the studies conducted in Germany, it is evident that the heat demand alone accounts for almost

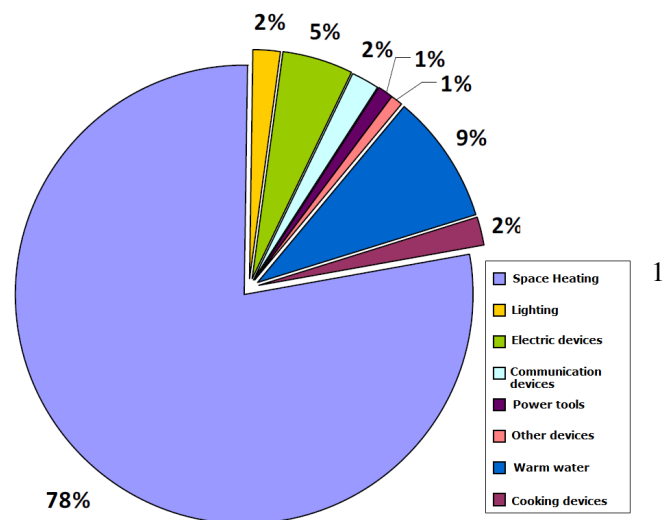


Figure 1. Energy usage in German Houses. Translated from (Krause, 2011)

78% of the total energy costs (Figure 1). Experimental studies regarding the implementation of μ -CHPs in residential clusters have established test subjects having high energy conversion efficiencies from 80% (Ren and Gao, 2010) up to 90% (VDI, 2013) and reduction in CO_2 emissions up to 42.5% (VDI, 2013). μ -CHP implementation studies done in Japan (Aki, 2007) also concluded that such systems are penetrating rapidly into the mainstream households and that it would have a positive stabilising effect on the electricity grid. Further, in combination with heat storage, they also provide cost efficient means of storing surplus energy from the grid locally (the surplus energy has to be first distributed). Distributed storage also involves analysis of various scenarios like peak energy supply and demand, duration of the day when the grid has surplus energy, the proportion in which it needs to be shared to various households and finally the methods of decentralised storage itself. Further, to analyse buildings as clusters at the regional level, not only should different buildings be incorporated into the model but also must the analysis focus on the perfor-

mance of different combinations of individual buildings when μ -CHP units are used as their primary heat source. The dynamic simulation of a total system (Figure 2) consisting of the gas grid, the power grid and the residential sector is essential to analyse the time dependant energy demand and supply behaviour of all the individual components. The development of relevant control systems on the other hand could control the interaction between the components which is important for the ground level implementation of such systems. A preliminary assumption could also be made that such storage methods would require lesser infrastructure and construction investment as most of the components are already constructed. However it needs to be further studied. This requires simu-

distributed storage strategy.

2 Modelling

In this study, modelling involves four steps as detailed below:

- Modelling the residential buildings
- Modelling the μ -CHP systems used inside the residential buildings
- Modelling the various auxiliary components and control systems
- Categorising buildings and re-dimensioning the components used in them according to their energy demands to develop a storage strategy

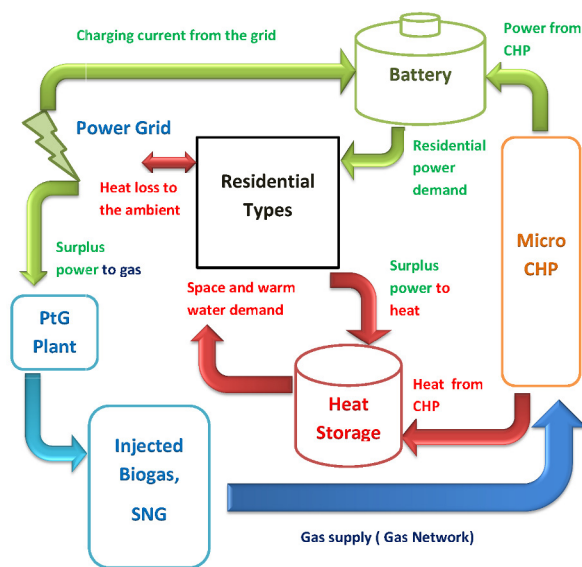


Figure 2. Concept diagram of distributed storage strategy

lations involving houses with μ -CHPs, their respective control systems and additional components. To analyse the buildings as a cluster, other parameters like geometric attributes of the buildings, details regarding the building materials and the energy usage patterns of the inhabitants also need to be analysed. Various models for calculation and simulation have already been suggested. The ESP-r model (Beausoleil-Morrison et al., 2012) presents an insight into the use of μ -CHP co-generation system combined with a storage unit in the household sector. In Modelica, μ -CHPs have been analysed in individual buildings in the residential sector with heat controlled and power controlled techniques (Stinner and Mueller, 2012). Further, libraries like the Modelica Buildings Library (Wetter, 2009) are available to analyse various systems possible in individual buildings using components for air based and water based heating systems, airflow controls and room to surrounding heat transfer models. The focus of this study however is not the in-depth analysis of the individual components themselves but how well the different aspects could be synchronised into a

An example of a house system is depicted in Figure 3. Here, the μ -CHP is the main heat production unit and from it, the heated water is transported to the tank where it is stored. The stored water is responsible for both space heating as well as satisfying the warm water demand. The space heating system includes the tank, the radiator and the pump in closed cycle while the warm water system is in open cycle where water is taken from the storage tank for tasks like showering or washing and the tank is later replenished to compensate for the water taken. The total warm water requirement per day is estimated for each household and a single tank is used. This means that the tanks used are over-dimensioned to accommodate enough water for satisfying both space heating as well as warm water requirements.

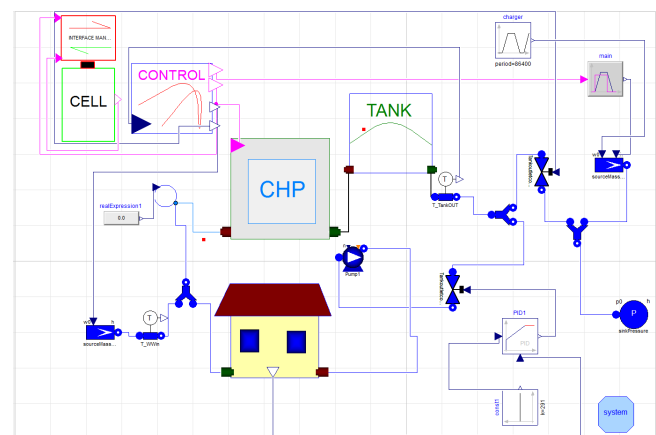


Figure 3. Screen shot of system model in Dymola.

2.1 Modelling the residential buildings

In the modelling of the residential buildings (Figure 4), when assumed that the hottest temperature is that of the heater surface and the coldest is the environmental temperature outside, heat is transferred from the heater to the

room and is then lost to the outside atmosphere through the walls. Here, either one or multiple walls may face the outside environment which is also taken into account while modelling the various house types. However, due to the complexities involved in the integration of various heating and ventilation subsystems and due to the fact that the focus of the study is more towards large scale distributed storage, detailed HVAC models like (Wetter, 2009) are not used. The flow control systems including the valves, tubes and heat exchanger models are either directly used from the Thermopower library (Casella and Leva, 2003) or modified from it. The modifications mostly lie in the calculation of the overall heat transfer coefficients. The heat transfer coefficients are calculated using the Nusselt number correlations for specialised cases as mentioned in the VDI Heat Atlas (VDI, 2010).

2.2 Heat Transfer

Heat is transferred from the surrounding to the house through the house walls, doors and windows in the direction of decreasing temperature. A room is conceptualised as having 6 wall surfaces each modelled with separate boundary conditions depending on the layout of the room. Additionally, there are doors, windows and openings. The geometry of all solid surfaces are defined using their respective areas and thickness values. Figure 4 shows the heat transfer through one such wall

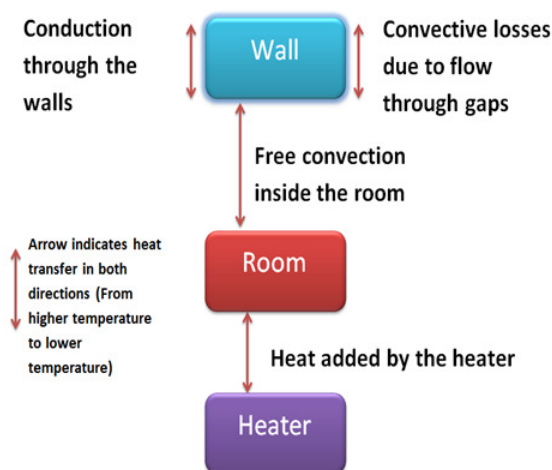


Figure 4. Concept of heat transfer used in building modelling

surface. As it depends on various factors like radiation, use of forced convection ventilation systems and the arbitrary opening and closing of doors and windows, the heat transfer correlations are difficult to be accurately calculated by incorporating all details. Therefore, the overall heat transfer coefficient used is taken directly from experimental studies (Causone et al., 2009) and (Defraeye et al., 2011) where the thermal properties of buildings were measured with and without furniture in nor-

mal usage conditions. This simplification also reduces the computing effort involved in calculating the individual contributions of various small heat sources. Arbitrary opening and closing of doors and windows is also accounted for. Here, the flow heat transfer is coupled with a Boolean signal which is active only as long as the doors or windows are open and the daily time of window and door opening and closing times are given as a time averaged value that repeats periodically during the simulation.

2.3 Modelling the μ -CHP systems

The μ -CHP unit is modelled using a heat exchanger concept. This assumes heat recovery from the μ -CHP system without modelling the combustion or working-cycle related details (or detailed electrochemical modelling in case of a fuel cell). This means that in contrast to the building models, modelling the μ -CHP unit involves leaving out certain geometric as well as performance related parameters. The simplified model however is validated using the in house lab facilities. Heat production is already assumed in the μ -CHP systems and only the heat recovery is modelled. In the μ -CHP unit, the power production is calculated directly using manufacturer provided efficiency equations without modelling associated components like generators or circuits. In this study, all μ -CHP units used are internal combustion engines based on the Otto Cycle as they were validated.

2.3.1 Validation and categorisation of the CHP Model

For the validation of the μ -CHP models, the GasPlus-Lab in Karlsruhe is used which is an in-house facility for experimenting on real time μ -CHP units. In-order to validate the simulation results, an experiment was carried out using a μ -CHP unit having similar parameters under the same initial and boundary conditions as in the simulation. The inlet flow rate of water into the μ -CHP heat recovery unit, the outlet flow rate as well as the inlet and outlet temperatures were measured in the experiment. The same procedure was imitated in the simulation model. All the geometric parameters that were possible to be measured in the experimental model were compared to the ones in the mathematical model and the rest were either assumed from manufacturer specifications or back calculated. The dynamic results of the test set-up as well as the simulation results were compared to validate the model (Figure 5). It has to be noted that due to the limitations of measuring equipment, the measurement intervals chosen were different from intervals used in the simulation. Therefore, the values missed in between were interpolated. Still, the deviation of the simulation results from the measured values stay within a range of ± 3 degC. This deviation may be attributed to the simplification of geometry in the mathematical model

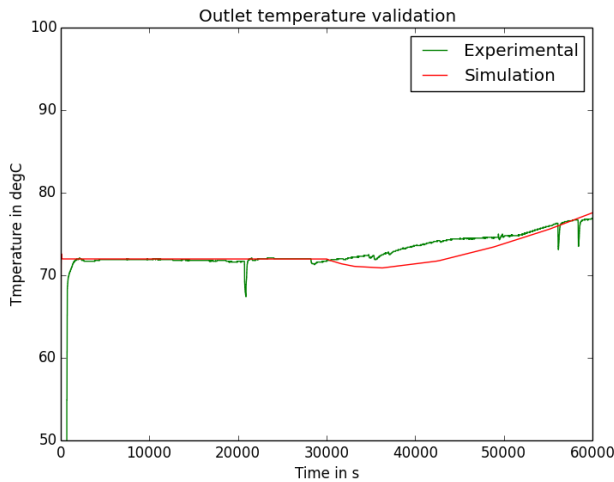


Figure 5. Measured CHP output fluid temperature for both the experimental and the simulation model.

or the use of simpler heat transfer coefficients.

3 Control system

One of the important aspects of this study has been the development of a control system that not only satisfies the heating requirements of the building which includes both space heating as well as warm water demand but also helps in the distributed storage of energy. The main challenge encountered in designing the control system was that all the following parameters defined below needed to be controlled simultaneously:

- Heating requirements of the room
- Warm water requirements of the room
- Storage of surplus electrical energy as heat
- Management of locally produced electrical energy (From μ -CHP)

As the requirements for each of the above mentioned aspects were different, a central system to control all the factors simultaneously could not be used. Instead, three separate control systems were developed for heating requirements, storage requirements and heat-electricity interface respectively.

3.1 Heat management strategy

Heat management essentially involves controlling both the space heating as well as hot water demand of the household. In the models used in this study, a single central tank is used for both. Two separate control systems are used for heat management:

- Hysteresis controller which switches the μ -CHP on and off to maintain the tank temperature levels.

- A continuous PID controller that controls the fluid flow rate into the radiator in-order to maintain a constant room temperature.

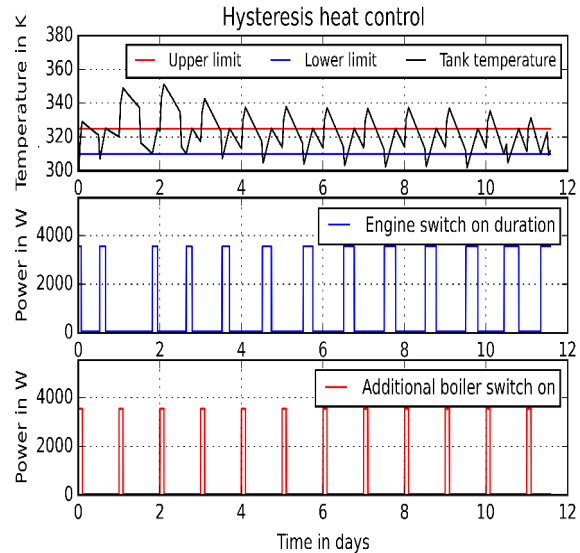


Figure 6. Hysteresis controller (top figure) switching on the main CHP unit (middle figure) with periodical working of the additional heat boiler (bottom figure).

Figure 6 shows the hysteresis controller that works between two temperature levels. For the fluid inside the tank, the upper cut-off and the lower cut-off limit temperatures are set initially and the hysteresis controller switches the μ -CHP on when the lower limit has been crossed and switches it off when the upper limit has been breached. Perfect mixing and a single uniform temperature is assumed for the fluid in the tank. It is to be noted here that the use of surplus power to heat boilers may cause the tank temperature to rise beyond the control value. (The additional boiler switches off however below the safety limit for fluid temperature inside the tank). Warm water demand per household per day is estimated from statistical data. From the main tank, the quantity of hot water corresponding to this heat demand is periodically released to a sink for the entire simulation period. The room temperature control uses a different approach. Here, the focus is to keep the room temperature constant at 18 deg C independent of outside weather fluctuations. This means that for space heating, continuous control is necessary. Therefore, a PID controller is used to control the inlet fluid flow into the radiator. Figure 7 shows the implementation of continuous control. When the temperature outside the room falls, heat is lost through the walls to the surroundings thereby lowering the room temperature. The PID controller then increases the inlet mass flow rate of hot fluid into the radiator till the room temperature gets back to the desired value.

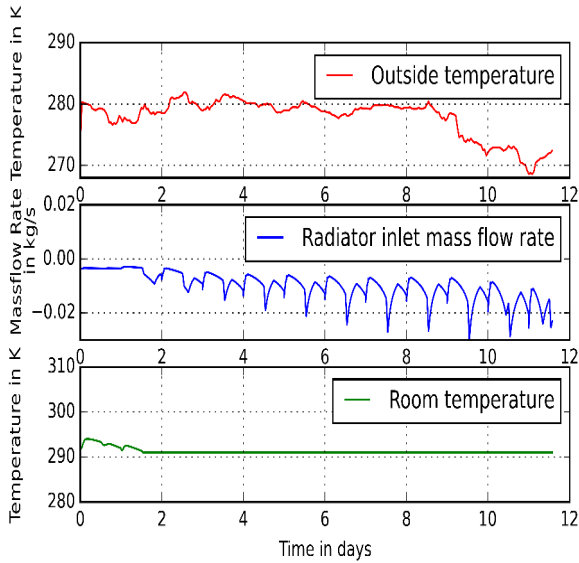


Figure 7. PID Controller flow regulation (middle figure) and the room temperature maintained as a result (bottom figure).

3.2 Power management strategy

As a heat prioritised strategy is used in this system, power management involves analysing scenarios where power produced in the μ -CHP system could be used directly or stored whenever it is available. The battery

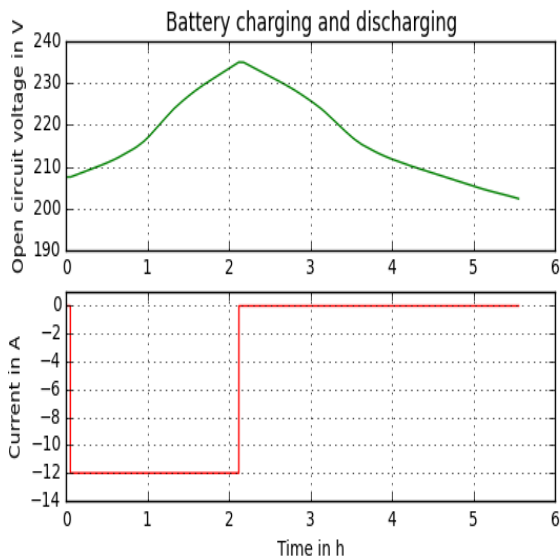


Figure 8. Charging and discharging cycle of battery.

model is a modified version of the energy storage model (Einhorn et al., 2011). An automated control system has not been implemented here which means that the control is based on a set of pre defined rules (defined below):

- The μ -CHP produces power and the battery needs to be charged

- The μ -CHP does not produce power but the battery needs to be charged
- The μ -CHP produces power but the battery is also full and in discharge mode
- The μ -CHP is off and the battery is full and in discharge mode

It is assumed that one of the above situations is present at any time during the simulation. Rules are defined separately for all the four cases and a summary is explained in Table 1. Figure 9 depicts the screen-shot of the electricity management system. A detailed dynamic model

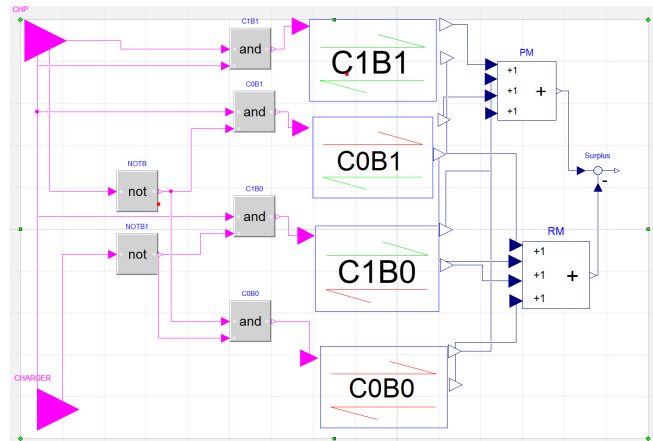


Figure 9. Screen-shot of the power interface system

incorporating all the household appliances has not been implemented for electricity management. The average electrical load for each household is calculated using a time averaged constant value based on experimental data and it is given as an input.

Table 1. Power management scenarios situation and reaction. C:CHP power production Boolean, B:Battery Charging Boolean

Situation	Reaction
C1B1	Battery charged with CHP power
C1B0	Power returned to the grid
C0B1	Battery charged from main supply
C0B0	House supplied by battery discharge

4 Energy distribution and storage using power to heat conversions

The concept of distributed power to heat has been depicted in Figure 10. In the first step, surplus energy at the grid level is proportioned dynamically and in real time based on the individual energy requirements of each house type (and also the number of houses in each type).

Subsequently, the individual proportions of distributed energy are stored locally using power to heat conversions. Finally, when the de-central units also have surplus power due to the working of the μ -CHP systems in heat prioritised mode, the surplus energy produced locally is either returned to the grid or used in other forms of storage (when both the grid and the houses have surplus energy).

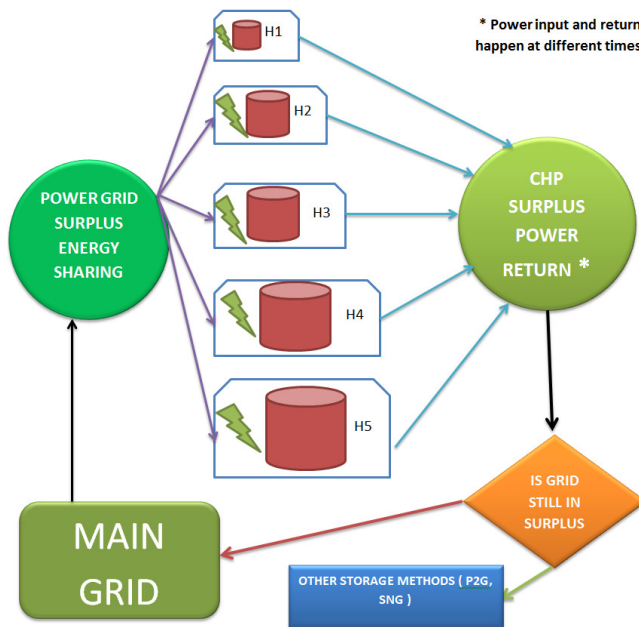


Figure 10. Concept of distributed energy sharing, localised storage and surplus energy return using house clusters (types H1 to H5).

4.1 Power to heat systems

As power to heat conversions are used for energy storage in individual houses, apart from the main μ -CHP unit, an additional boiler is also integrated into the system which works only when the main grid has surplus power. To analyse power to heat systems, it is assumed that a part of the surplus energy available in the grid has already been correctly proportioned and transmitted to the house. The storage tank is designed to store heat both from the μ -CHP unit and the surplus energy heater simultaneously. Figure 11 explains the concept further. Here, the duration of μ -CHP operation is controlled using the hysteresis controller but additional heat energy is stored whenever it is available as long as the safety limit temperature of the systems are not breached (in this particular study, a boolean pulse was used to denote the availability of surplus grid energy at regular daily periodic intervals). As a result, the number of times the μ -CHP unit has to switch on daily is also reduced. This increases the overall life of the μ -CHP unit which is an important justification for its purchase initially. The experience from real life tests conducted at the GasPlusLab in Karlsruhe indicate that intermittent switching on and off of μ -CHP units is not

desirable as it may lead to higher repair and maintenance costs. Additionally, in a grid connected scenario where μ -CHP units are operated part time and other renewable energy sources are used when they in turn are cheaper to produce, it is also not necessary to operate the de-central units all the time if an overall optimised operation is desired.

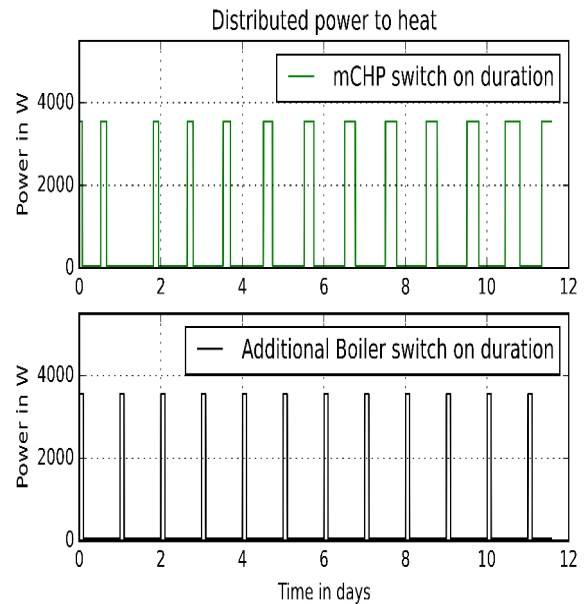


Figure 11. Simultaneous switch on of extra boiler and main μ -CHP unit for power to heat storage.

5 First results: Simulation of the household types

For decentralised storage systems, the accurate real time distribution of surplus energy in the grid also involves the accurate real time estimation of the demand arising from various households. To develop a decentralised storage strategy, two possible scenarios are analysed in the simulation of houses with μ -CHP systems:

1. Simulation of houses with μ -CHP systems using an additional heater with on/off control for surplus energy storage
2. Simulation using only μ -CHP systems that could be operated dynamically between part load and full load mode depending on storage requirements

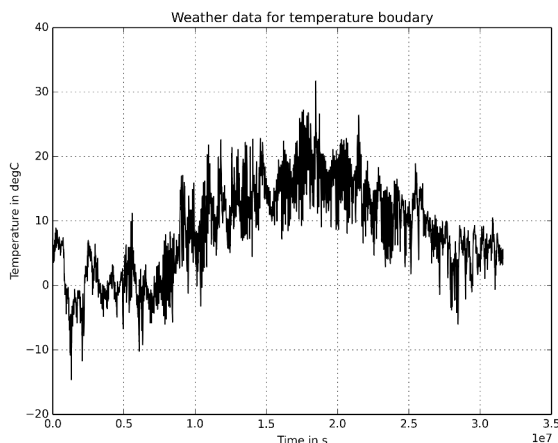
5.1 Test case 1: Simulation of house types using μ -CHP systems with additional boiler

The initial set of simulations were carried out for a cluster with all the five household types. All the household types were given temperature boundary conditions based

Table 2. Categorisation of building types

House name	H1	H2	H3	H4	H5
No: Inhabitants	2	3	4	5	7
Floor Area [m^2]	110	160	320	430	900

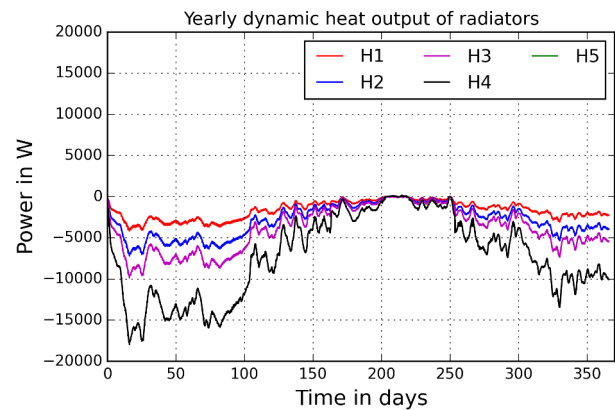
on measured weather data available for Karlsruhe. The thickness of the insulation materials and the materials used for construction (doors, windows, walls and insulation) were kept uniform for all the house types. This means that only the dimensions of the different house types differ and they are defined in Table 2. The control systems are the same as described in earlier sections. The simulation is done for a year with location based measured weather data (The dynamic weather fluctuations were incorporated using temperature boundary conditions, see Figure 12). Figure 13 shows the yearly dynamic output of the respective in-house radiators when a constant temperature is maintained in the room. The dy-

**Figure 12.** Measured temperature at hourly intervals between August 2013 and 2014.

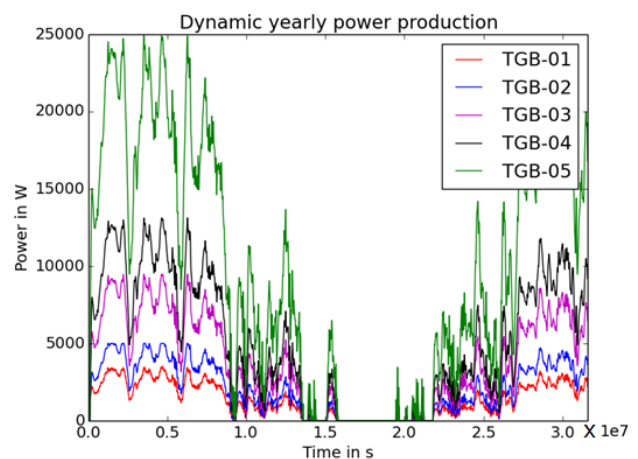
amic estimation of the heat load defined in Figure 13 is satisfied using heat recovery from a CHP system. This dynamic estimation of the house energy load is performed only as a capability demonstration. The results could be made more accurate if more details regarding construction materials of the actual buildings, the energy usage behaviour of the inhabitants, the actual geometries of the buildings, the number of buildings in each type and location based weather data are incorporated into the simulation.

5.2 Test case 2: Simulation part-load capable μ -CHP systems

The dynamic operation of the μ -CHP system between part load and full load depending on heating and stor-

**Figure 13.** The variation of heating load among different types of houses.

age requirements is also a possibility in the future. Although presently available residential μ -CHP systems do not have such a capability, in the future, if such systems are available, it would offer the option to control the entire decentralised storage system dynamically. Such systems also offer advantages in buildings with multiple families and multiple users with different energy usage patterns as real-time increase or decrease of output would be possible according to the varying load. For this reason, a conceptual model is also created using a μ -CHP system that could dynamically work between part load and full load. This is accomplished by replacing the on-off switch of the μ -CHP system using a continuous PID controller. The initial results of simulations involving part load operation is depicted in Figure 14.

**Figure 14.** Dynamic Part load operation of CHP units showing variation in power production. (TGB Indicates buildings using part load capable μ -CHP systems)

6 Future and prognosis

As shown in Figure 2, the availability of heat storage tanks in residential applications as well as batteries for electrical storage could be used for short term and localised storage of either heat, electricity or both. But a larger strategy involving the gas grid and the power grid could offer other potential advantages like integration of different renewable energy sources and fewer infrastructure requirements.

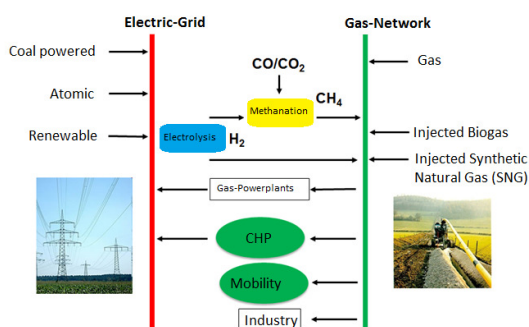


Figure 15. Planned future synchronisation between the electrical and gas networks.

Integration of other renewable technologies

One possibility of implementing the distributed surplus energy storage is by real time distribution of energy using the power grid and localised storage (Power-to-Heat or Batteries). However, there is also another possibility for local energy production which could potentially be largely (or even completely) independent of the power grid. This requires new technologies like Power to Gas (PtG) systems (Jentsch, 2015) that initially convert surplus power to H_2 and inject it directly into the gas-grid as long as the procedures involved comply with the respective gas norms. Another possibility is to use technologies like methanation to convert H_2 to CH_4 which could then be injected into the grid. Injection of Bio-Gas into the main gas grid is also presently being done on a large scale. The most important motivation is that as long as gas based CHP systems are used in the residential sector, it could be efficiently complemented by gas based storage technology at the grid level. This is because the gas converted using Power to Gas or methanation technologies could be stored in normal containers without huge infrastructure requirements longer than any conventional electrical energy storage technology and the gas is also easily transportable through the gas network to the houses when heat or power demand arises locally. Studies at DVGW are presently focussed on the optimal quantity of H_2 that could be safely injected into the main gas grid without affecting the calorific values of the transported gases (Burmeister et al., 2012) and also the technologies and energy requirements for methanation and optimal injection of Biogas into the gas grid. However, it has to be emphasised that there is no single storage solution that could satisfy all the stochastic storage requirements si-

multaneously. The integration of renewable energy and the development of associated storage systems may require a combination of many strategies like power to heat conversions, power to gas production, methanation, distributed energy storage or other storage methods. The study carried out is an important capability demonstrator and an initial step in the overall strategy of integrating the gas grid, the electricity grid, the residential sector and probably the mobility sector too into an interacting unit where members have a symbiotic relationship. More importantly optimising their interactions in the future could possibly be one of the more realistic options for large scale energy storage without having huge infrastructure demands.

References

- Hirohisa Aki. The penetration of micro CHP in residential dwellings in Japan. *2007 IEEE Power Engineering Society General Meeting, PES*, pages 1–4, 2007. ISSN 1932-5517. doi:10.1109/PES.2007.385625.
- Ian Beausoleil-Morrison, Michaël Kummert, Francesca MacDonald, Romain Jost, Timothy McDowell, and Alex Ferguson. Demonstration of the new ESP-r and TRN-SYS co-simulator for modelling solar buildings. *Energy Procedia*, 30:505–514, 2012. ISSN 18766102. doi:10.1016/j.egypro.2012.11.060.
- Frank Burmeister, , Jens Senner, Janina Brauner, and Rolf Albus. Potenziale der Einspeisung von Wasserstoff ins Erdgasnetz – eine saisonale Betrachtung. *Energie*, 4:52–57, 2012.
- Francesco Casella and Alberto Leva. Modelica open library for power plant simulation: design and experimental validation. *Proceedings of the 3rd International Modelica Conference*, pages 41–50, 2003. URL <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Modelica+open+library+for+power+plant+simulation+:+design+and+experimental+validation#0>.
- Francesco Causone, Stefano P. Corgnati, Marco Filippi, and Bjarne W. Olesen. Experimental evaluation of heat transfer coefficients between radiant ceiling and room. *Energy and Buildings*, 41(6):622–628, 2009. ISSN 03787788. doi:10.1016/j.enbuild.2009.01.004.
- Thijs Defraeye, Bert Blocken, and Jan Carmeliet. Convective heat transfer coefficients for exterior building surfaces: Existing correlations and CFD modelling. *Energy Conversion and Management*, 52(1):512–522, 2011. ISSN 01968904. doi:10.1016/j.enconman.2010.07.026. URL <http://dx.doi.org/10.1016/j.enconman.2010.07.026>.
- M. Einhorn, F. V. Conte, C. Kral, C. Niklas, H. Popp, and J. Fleig. A Modelica Library for Simulation of Electric Energy Storages. *Proceedings 8th Modelica Conference*, pages 436–445, 2011. doi:10.3384/ecp11063436. URL http://www.ep.liu.se/ecp_article/index.en.aspx?issue=63;article=48.

- Mareike Jentsch. Wirtschaftlicher Einsatzbereich von PtG - Energiespeichern im erneuerbaren Stromversorgungssystem. In *Internationale Energiewirtschaftstagung an der TU Wien IEWT 2015*, pages 1–12, Vienna, 2015.
- H Krause. Krause H, Erler F. Bewertung der Energieversorgung mit leitungsgebundenen gasförmigen Brennstoffen im Vergleich zu anderen Energieträgern (Teil I), AP4: ,Freiberg, Germany, Nachfragestruktur, Bedarfs- und Bestandsanalyse. (G 5/04/09-TP1-C), 2011.
- Hongbo Ren and Weijun Gao. Economic and environmental evaluation of micro CHP systems with different operating modes for residential buildings in Japan. *Energy and Buildings*, 42(6):853–861, 2010. ISSN 03787788.
- Sebastian Stinner and Dirk Mueller. Thermal Simulation of Power-Controlled Micro-CHP Systems for Residential Buildings. *Proceedings 9th Modelica Conference*, pages 935–940, 2012. doi:10.3384/ecp12076935. URL http://www.ep.liu.se/ecp_article/index.en.aspx?issue=76;article=97.
- VDI. VDI Heat Atlas. Technical report, 2010. URL <http://www.springer.com/de/book/9783540778769#aboutBook>.
- VDI. Statusreport 2013 Mikro-Kraft-Wärme-Kopplungsanlagen. Technical report, 2013. URL <http://www.vdi.de/presse/artikel/statusreport-2013-zu-mikro-kraft-waerme-kopplungsanlagen/>.
- Michael Wetter. Modelica Library for Building Heating, Ventilation and Air-Conditioning Systems. *Proceedings 7th Modelica Conference*, pages 393–402, 2009. doi:10.3384/ecp09430042. URL http://www.ep.liu.se/ecp_article/index.en.aspx?issue=043;article=44.

Test of Basic Co-Simulation Algorithms Using FMI

Kosmas Petridis¹ Christoph Clauß²

¹Robert Bosch GmbH, Corporate Sector Research and Advance Engineering, Robert-Bosch-Campus 1, 71272 Renningen, Germany, kosmas.petridis@de.bosch.com

²Fraunhofer IIS EAS, Zeunerstrasse 38, 01069 Dresden, Germany, christoph.clauss@eas.iis.fraunhofer.de

Abstract

Since the FMI technology gains ground in industrial environment, the demand for robust co-simulation increases. In a master-slave concept the master algorithms define the quality of a co-simulation whereas the properties of the coupled FMUs for co-simulation restrict the variety of possible master algorithms. In this paper an existing experimental master tool with three basic master algorithms was improved to support FMI 2.0 as well as 1.0. For testing more than 20 Modelica examples were developed from which FMUs for co-simulation were generated by established simulation tools (e.g., Dymola, SimulationX). The examples demonstrate differences of the three master algorithms. Recommendations for tearing as well as improving the master algorithms are given.

Keywords: Co-Simulation; FMI; master algorithm;

1 Introduction

Nowadays simulation is of crucial importance in the development of mechatronic and cyberphysical systems. The main characteristic of such systems is that they consist of components of different physical domains like hydraulic, mechanic, electronic, and software. Through the strong coupling between the components the isolated investigation of single components is not sufficient. In fact the overall system has to be investigated. This means that we need to simulate the complete system. In general, the components are modelled and simulated in different established simulation tools. One commonly used method to simulate the complete system is co-simulation which can be classified into two types: the direct coupling between tools and the export and import of the simulation model into the other tool. To do this, there exist a lot of proprietary commercial and self-developed solutions but all of them are only applicable on a limited number of tool combinations. In addition these solutions need a high effort in maintenance because of the proprietary interface to the different simulation tools. A further disadvantage of these solutions is that the algorithms used for the coupling are strongly coupled with the interface. In addition usually only standard algorithms based on a constant macro

step size are used. To avoid these limitations the Functional Mock-up Interface (FMI) was developed as an interface standard which allows the exchange and co-simulation of models. The standard allows the use of different coupling algorithms within the same interface. The coupling algorithms themselves are not part of the standard. Because of the increasing number of simulation tools, which support this standard, and the need from an industrial point of view (Bertsch et al, 2014) FMI represents a promising industry standard for model exchange.

2 Co-Simulation in Industrial Environment

One example where co-simulation is used to analyze the system is the simulation of injection valves (Petridis, 2013). The following physical domains are simulated with different simulation tools:

- Hydraulics and mechanics
- Electromagnetics and power electronics

Numerous additional examples for co-simulation like the simulation of high-pressure pumps, breaking systems, etc. exist.

Based on these applications we determined the following coupling cases:

- Simulator specific model with one imported FMU
- Simulator specific model with more than one imported FMU
- Software in the loop (SIL) platform with control algorithms and one or more FMU plant models

Thereby the type of coupling can be distinguished by:

- Coupling in one direction (see Figure 1) or with feedback (see Figure 2). The last one is also known as cycle.
- Analog coupling quantities (displacement, force, etc.) or discrete coupling quantities (sensor or actor signals)

The different simulation models can have the properties:

- Algebraic system without solver
- Differential or differential algebraic equation including solver (based on constant or variable solving step size) or without solver

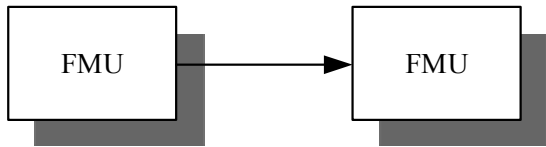


Figure 1. One directional coupling between two FMUs.

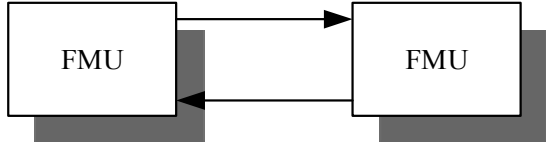


Figure 2. Coupling with feedback between two FMUs.

The described coupling configurations are an incomplete snapshot based on the current co-simulation applications. But it is a very useful orientation to determine the requirements on coupling algorithms.

3 Basic Co-Simulation Algorithms

Assume, m simulators S_i , $i = 1(1)m$, are to be coupled. Sometimes, such a simulator S_i is called “slave”. Typically, it is “packaged” in a FMU. These simulators are assumed to exchange altogether n coupling variables $x_j(t) \in R^1$, $j = 1(1)n$, $t \in [0, T]$, which are time dependent. Some of the coupling variables are input variables for a simulator S_i , other coupling variables are output variables. It is assumed that no input variable is output variable of the same simulator. $[0, T]$ is the time interval to be simulated. If $x = (x_1, x_2, \dots, x_n)^T \in R^n$ is the vector of all coupling variables then the call of each single simulator can be described by $x = Q_i S_i(P_i x)$.

P_i is a $(p_i \times n)$ -matrix with one “1” at each row and all other entries identical zero. It denominates which coupling variable is the input of the simulator S_i . S_i has p_i input variables. Q_i is a $(n \times q_i)$ -matrix with one “1” at each column and all other entries identical zero. It denominates which coupling variable is the output of the simulator S_i . S_i has q_i output variables. Since each coupling variable is output of exactly one simulator, $\sum_{i=1}^m q_i = n$ holds. Furthermore, the $(n \times n)$ -matrix $Q = (Q_1, Q_2, \dots, Q_m)$ has exactly one “1” in each column and in each line. Since no input variable is assumed to be an output variable of the same simulator $P_i Q_i = \theta$ holds with θ being a $(p_i \times q_i)$ -zero-matrix. The matrices P_i and Q_i describe the connection graph, that means how the input variables and output variables of each simulator are connected.

To illustrate the notation of coupling, the following example is given in Figure 3. We have $m = 2$ simulators S_1 and S_2 with $n = 4$ coupling variables $x = (x_1, x_2, x_3, x_4)^T$. S_1 has $p_1 = 1$ input variables and $q_1 = 3$ output variables. Thus the dimension of P_1 is (1×4) and the dimension of Q_1 is (4×3) . The

matrices are $P_1 = (0 \ 0 \ 0 \ 1)$ and $Q_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$.

S_2 has $p_2 = 3$ input variables and $q_2 = 1$ output variables. Thus the dimension of P_2 is (3×4) and the dimension of Q_2 is (4×1) . The matrices are $P_2 =$

$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ and $Q_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$. The input variables of

S_1 are described with $P_1 x = x_4$ and of S_2 with $P_2 x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$. The output of S_1 is given by the (3×1) -vector

$S_1(P_1 x)$ and of S_2 by the (1×1) dimensional $S_2(P_2 x)$. The multiplication of these terms with the corresponding matrix Q

$$Q_1 S_1(P_1 x) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} S_1((x_4))$$

$$Q_2 S_2(P_2 x) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} S_2 \left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \right)$$

corresponds to a mapping of the outputs of each simulator to the coupling vector x .

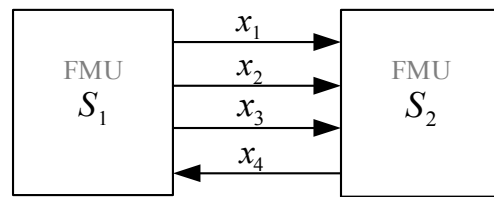


Figure 3. Example to describe the notation

Using this notation the task of the coupled simulation can be described as the following task: Find x^* which solves:

$$x^* = \sum_{i=1}^m Q_i S_i(P_i x^*) \tag{1}$$

In general, this equation is a nonlinear equation in the space of time dependent functions. All solution methods which are available to solve nonlinear equations should be checked to solve this equation.

First fixed point iteration methods are possible which take equation (1) “as it is”. There are several approaches. With k being the iteration index, and $x^0(t) = (x_1^0(t), \dots, x_n^0(t))$, $t \in [0, T]$ the initialization of coupling variables, the Gauss-Jacobi method can be characterized by

$$x^{k+1} = \sum_{i=1}^m Q_i S_i(P_i x^k) \tag{2}$$

In this case the simulators S_i can operate in parallel, since all simulators access to the same vector of coupling variables x^k .

Another variant is the Gauss-Seidel method (GSM) which needs an a priori defined calling sequence $r = (r_1, \dots, r_m)$ of the simulators. Each simulator uses the results of the already called simulators. One iteration step is finished if all simulators have been called. The Gauss-Seidel method can be summarized by

$$\xi^1 := x^k$$

$$\xi^{i+1} := Q_{r_i} S_{r_i}(P_{r_i} \xi^i) + \xi^i - Q_{r_i} Q_{r_i}^T \xi^i \quad (3)$$

$$i = 1(1)m$$

$$x^{k+1} := \xi^{m+1}$$

The sequence r is defined by analyzing the matrices P_i and Q_i which is the same as to analyze the connection graph. The sequence shall be chosen such that as many as possible input coupling variables are “updated” before the simulation of each slave simulator.

A special case of the Gauss-Seidel method takes one iteration ($x^0 \rightarrow x^1$) only. That means that no “true” iteration takes place. This method is sufficient if a sequence r of simulator calls can be found at which each simulator takes input values only which are outputs of before called simulators.

Equation (1) can be reordered into

$$0 = x^* - \sum_{i=1}^m Q_i S_i(P_i x^*) =: x^* - Sx^* \quad (4)$$

To find a “root” of (4) Newton like methods can be applied, e.g. the classical Newton-Raphson method (NRM):

$$x^{k+1} = \left(\frac{\partial S}{\partial x} - I\right)^{-1} \left(\frac{\partial S}{\partial x} x^k - Sx^k\right) \quad (5)$$

In addition, for all groups of methods (2), (3), (5) many modifications are known (Schwetlick, 1979), e.g. the introduction of damping methods which limit large changes of the solution between two iterations.

So far all of these methods are applied to the complete functions (t) , $t \in [0, T]$. This results in waveform relaxation methods and waveform Newton methods respectively which operate with functions within a function space on the time interval $[0, T]$.

Since FMI is not designed to exchange function space variables but values at certain time points the time interval is segmented into subintervals (communication intervals) $[0, T] = \cup [t_c, t_{c+1}]$. Each time t_c is a communication point at which the simulation of all slave simulators S_i is stopped for the exchange of values between the master and the slave simulators. $h_c = t_{c+1} - t_c$ is the communication step size (macro step size) between communication points which can be variable or constant. The above described task of simulator coupling (1) is solved for each communication interval $[t_c, t_{c+1}]$. The simulation of a slave simulator

S_i within a communication interval is performed by the FMI doStep function. Both the methods (2) and (5) need repeated simulations of communication intervals, the “FMUState” must be stored (GetFMUState) and used again (SetFMUState).

The presented approaches yield a high variety of methods which have to be chosen depending on the properties of the simulation task and the restrictions of the FMUs to be coupled. A master should offer many coupling algorithms to be able to choose the best suitable one for a special coupling task. General criteria for the quality of master algorithms are the performance (it touches questions like this: are slave simulations repeated within communication intervals?), the correctness of the results (it touches questions of stability with respect to the communication step size) as well as the robustness (is an algorithm suitable for a large class of simulation tasks?). The choice of an ideally adapted master algorithm is an active field of research.

4 Tool for Testing Co-Simulation Algorithms

In (Bastian et al, 2011) a prototypical master tool for co-simulation is presented which was developed in the MODELISAR project by Fraunhofer IIS EAS Dresden. The aim was to investigate basic co-simulation algorithms while the Functional Mockup Interface was being created. The prototypical master tool coupled slaves written in C via a provisional interface which possessed the main functionality of FMI 1.0.

Recently, this “EAS master tool” was improved by supporting both the FMI 1.0 and FMI 2.0 for co-simulation. Furthermore, a graphical user interface for convenient handling was added. Currently, there are efforts to unify the interface of controlling master algorithms via an XML-File. These efforts are also supported in a preliminary way. This improved EAS master opens the opportunity to thoroughly test its master algorithms, since examples can be modeled easily using Modelica, and exported as FMU for co-simulation with commercial simulation tools (e.g., Dymola, SimulationX). These FMUs can be coupled via the EAS master tool.

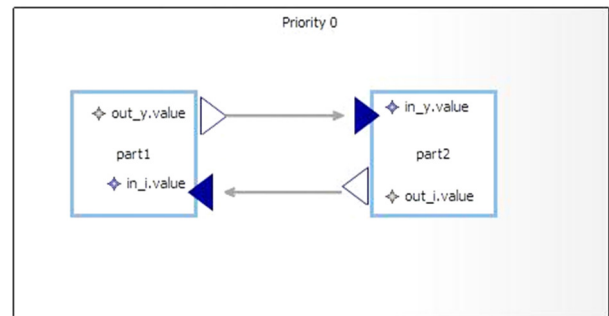


Figure 4. Graph window of the master GUI

The EAS master tool is controlled by a configuration text file with the following structure:

```
nval          2
nsim          2
tstart       0.0
tend         10.0
tstepmax     0.001
tstepstart   0.001
MasterMode   2
MasterDebug  2
OutputGnuplot 1
it_max_steps 100
it_tol_abs   1.0E-6
it_tol_rel   1.0E-4
simulator    0  fmus/part1.fmu
simulator    1  fmus/part2.fmu
graph #val #sim -1(out)/1(in) valueref
0 0 -1 r part1.out_y.value
0 1 1 r part2.in_y.value
1 1 -1 r part2.out_i.value
1 0 1 r part1.in_i.value
end
priority #sim priority
0 0
1 0
end
cycles #prior 0(no)/1(yes)
0 1
end
```

The configuration file contains the number of coupling variables (*nval*), the number of FMUs (*nsim*), the simulation time interval ([*tstart*, *tend*]), the communication step size (*tstepmax*, *tstepstart*), the chosen master algorithm (*mastermode*), some numerical parameters, the paths to the FMUs, the connection graph, and information on directions (*priority*) and cycles in the graph. The graphical user interface generates the configuration file.

The “EAS master tool” comprises the following algorithmic approaches:

- Constant communication step size
It is user-defined before simulation. Though variable communication step size basing on Richardson extrapolation was investigated successfully (Schierz et al, 2012; Schierz, 2013) it is not yet integrated into the tool.
- Sequence of calling the simulators
The sequence $r = (r_1, \dots, r_m)$ of calling the simulators is not yet automatically generated. It is to provide by the user.
- Gauss-Seidel method (3)
It is applied to each communication interval. This method requires the FMUs to be able to repeat the simulation of communication intervals (repeated *doStep* calls).
- Gauss-Seidel method (3) with one iteration step (GS1)

This simplified algorithm comes to a result in any case. But if there are cyclic dependencies the result may be no solution. Provided that the dependency sequence *r* is correct this method finds a solution if there are no cyclic dependencies.

- Newton-Raphson method (5)
This method requires repetitions of simulating communication intervals (repeated *doStep* calls over the same communication interval). The Jacobian is calculated applying difference quotients which needs additional simulations. Jacobians delivered by the FMU are not yet evaluated.
- Directed graph with included cycles
The dependencies between the simulators form a graph. The tool supports a unidirectional graph with included cycles. The iterating methods specified above can be restricted to the cycles, whereas GS1 is applied to the non-cyclic parts generally.

The “EAS master tool” solely interprets the information on the slave simulators given by FMI. Further information on the solution method used within the FMU is not exploited.

5 Application Test Examples

The „EAS master tool“ was applied to a lot of small test examples which address more or less different difficulties or aspects of co-simulation. Each example was first modeled using Modelica tools (Dymola, SimulationX) without partitioning to generate the reference solution. Second, the example was split, and each part was exported as an FMU. Using the master tool the FMUs were coupled, and the example was simulated using the three above mentioned basic algorithms.

Table 1 gives an impression of the behavior of the three basic algorithms. The following subsections present four of the examples (row 7, 10, 5, and 23 in

Table 1) in more detail. Finally, the last subsection collects some recommendations for succeeding co-simulations.

Table 1. Algorithm test results

	<i>Addressed Purpose</i>	<i>C</i> <i>y</i> <i>c</i>	<i>F</i> <i>M</i> <i>U</i>	<i>G</i> <i>S</i> <i>l</i>	<i>G</i> <i>S</i> <i>M</i>	<i>N</i> <i>R</i> <i>M</i>
1	straightforward system with correct calling sequence, no difficulties	0	7	Y	Y	Y
2	linear system, diagonally dominant matrix, iterations needed	1	5	d	Y	Y
3	digital and analog variables, events	1	3	d	d	d

4	like 2, but not diagonally dominant	1	5	w	w	Y
5	like 2, time dependent matrix, which loses diagonal dominance, see section 5.3	1	5	w	w	Y
6	precision test, “too large” communication step size	0	3	d	d	d
7	precision test, iterations needed, see section 5.1	1	3	d	Y	Y
8	precision test, iterations needed	1	3	d	Y	Y
9	nonlinear equation, iteration needed,	1	3	w	Y	Y
10	changing signal flow due to varying resistors, but fixed directions for FMI, see section 5.2	1	3	w	e	Y
11	like 10, but changed fixed directions	1	3	w	e	Y
12	like 10, further changed fixed directions	1	3	d	Y	Y
13	like 10, further changed fixed directions	1	3	w	e	Y
14	extended circuit with changing signal flow, but fixed directions for FMI	1	6	d	Y	Y
15	like 10, jumping input variables, events should be hit	1	3	w	e	Y
16	Rossler DAE, large simulation interval, iterations needed	1	3	w	d	d
17	retarding DAE	0	2	Y	Y	Y
18	like 5, other time dependent matrix, which loses diagonal dominance	1	5	w	w	Y
19	linear equation, not contractive matrix	0	2	d	d	Y
20	like 7, extended, test of initialization	1	4	e	e	e
21	higher index problem due to wrong signal flow direction; correct, that no FMU exportable	1	2	e	e	e
22	like 21, suitable direction	1	2	Y	Y	Y
23	“nearly” higher index problem, like 21, see section 5.4	1	2	d	d	Y

Legend: Cyc – number of cycles, FMU – number of FMUs, GS1 – Gauss-Seidel method with one iteration, GSM – iterating Gauss-Seidel method, NRM – Newton-Raphson method, Y – correct, d – small differences compared with unpartitioned reference solution, w – completely wrong result, e – error, or simulation fails, or FMU not exportable. The difference between “d” and “w” is estimated subjectively to give a rough impression whether the calculated solution is “far away” or “near” the correct solution.

5.1 Precision Test Example

The equations according to Table 2 are segmented such that the equations of each row in the table are simulated with their own simulator within an FMU. The columns “In” and “Out” describe the coupling variables.

Table 2. Equations of the precision test example

In	Equations	Out
x_2	$x_1 = -x_2$	x_1
x_1	$\frac{\partial x_2}{\partial t} = x_1, \quad x_2(0) = 1$	x_2
x_2	$e^{-t} - x_2 = y$	y

This example (row 7 in Table 1) is designed such that $y(t)$ is zero. Therefore, the magnitude of y is an indicator of precision of the numerical solution. All three implemented methods calculate the correct result. Both GSM and NRM are similar precise (Figure 5), but GS1 is more inaccurate (Figure 6). The communication step size was 0.1.

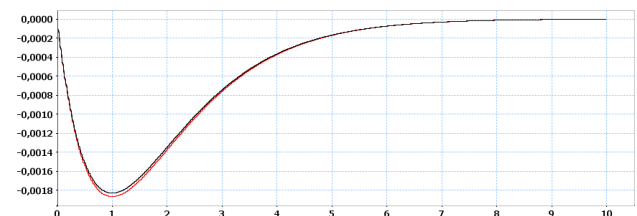


Figure 5. $y(t)$ calculated by GSM and NRM

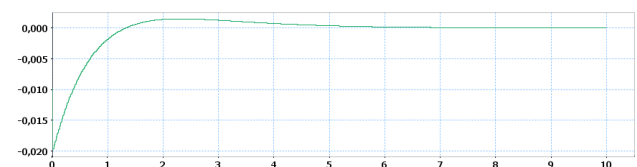


Figure 6. $y(t)$ calculated by GS1

5.2 Varying Resistors

This example (Figure 7, row 10 of Table 1) from the electronic domain is designed such that the voltage v_i alternately depends on x_1 or x_2 . The reason for this behavior is the variation of the resistances of $var1$ and $var2$ (Figure 8).

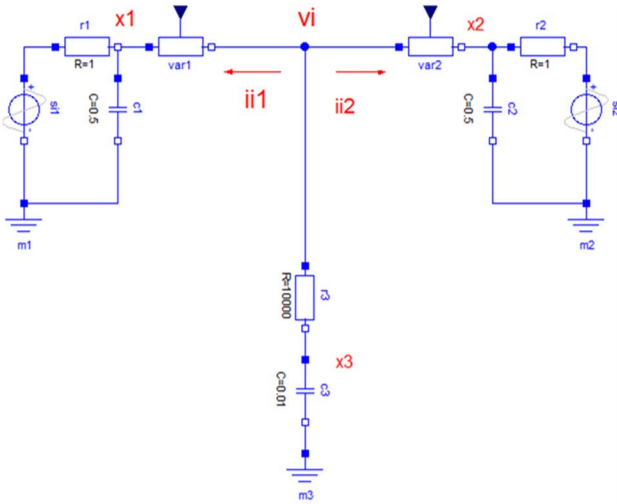


Figure 7. Circuit with varying resistors

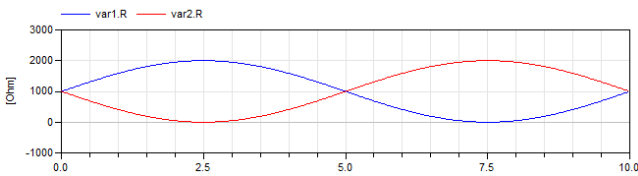


Figure 8. Varying resistances

Similar to Table 2 the equations of this example are presented in Table 3.

Table 3. Equations of the varying resistors example

In	Equations	Out
vi	$0.5(\partial x_1/\partial t) + x_1 - ii_1 - \sin(3\pi t) = 0$ $(vi - x_1)/(1000 \sin(0.2\pi t) + 1001) = ii_1$	ii_1
vi	$0.5(\partial x_2/\partial t) + x_2 - ii_2 - \sin(2\pi t) = 0$ $(vi - x_2)/(-1000 \sin(0.2\pi t) + 1001) = ii_2$	ii_2
ii_1, ii_2	$0.0001(x_3 - vi) + 0.01(\partial x_3/\partial t) = 0$ $ii_1 + ii_2 - 0.0001(x_3 - vi) = 0$	vi

Because of the varying resistances no unique exchange direction of coupling variables is possible. Therefore, both GSM and GS1 do not converge. Only NRM calculates a correct result (Figure 9). The trajectory is not quite smooth due to a large communication step size of 0.1.

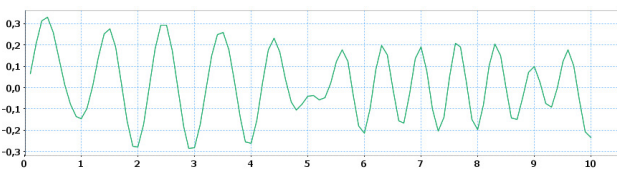


Figure 9. Coupling variable $vi(t)$

5.3 Linear System of Equations

In the following linear system of equations (row 5 of Table 1) the matrix varies depending on the time. Therefore, a similar behaviour like in Section 5.2 can be observed.

Table 4. Linear system of equations

In	Equations	Out
	$r_1 = 1, r_2 = t, r_3 = 1$	r_1, r_3, r_3
x_2, x_3, r_1	$3x_1 + (0.1 + t)x_2 + 0.2x_3 = r_1$	x_1
x_1, x_3, r_2	$0.1x_1 + 3x_2 + (0.1 + t)x_3 = r_2$	x_2
x_1, x_2, r_3	$(0.1 + t)x_1 + 0.2x_2 + 4x_3 = r_3$	x_3
x_1, x_2, x_3	$x_1 + x_2 + x_3 = y$	y

Five FMUs are coupled according to Table 4. Both GSM and GS1 do not converge, since due to the time dependence the fixed point iteration is not contractive for increasing t . The NRM calculates the correct result (Figure 10).

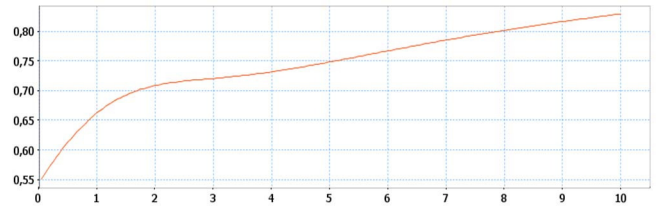


Figure 10. Result variable $y(t)$

5.4 Resistor-Capacitor-Circuit

This example (row 23 in Table 1) from the electric domain is a simple resistor-capacitor-circuit where the resistor is divided into two parts. The equations are allocated to two FMUs according to Table 4.

Table 4. Equations of the Resistor-Capacitor-Example

In	Equations	Out
i	$i(1 - R_{part2}) = y - u$ $u = \text{if } t < 2 \text{ then } 0 \text{ else if } 4 \leq t \text{ then } 2 \text{ else } 4$	y
y	$i \cdot R_{part2} = x - y,$ $-i = \partial x/\partial t, x(0) = 2$	i

The DAE index of the second FMU is one. But the smaller R_{part2} becomes the “closer” the index gets to two, which occurs if R_{part2} is zero. Therefore, difficulties in the coupled simulation arises, if R_{part2} is small. The corresponding result with $R_{part2} = 0.001$ obtained by the generating tool without partitioning is shown in Figure 11.

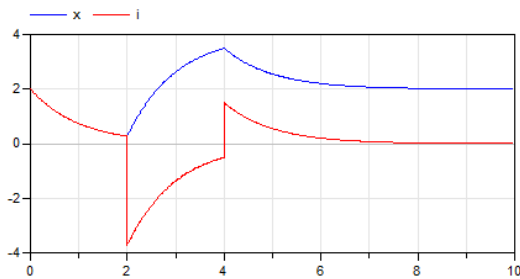


Figure 11. Reference solution Resistor-Capacitor-Circuit

For this example GSM fails. GS1 produces an extremely increasing result. The result of Newton's method is not quite exact due to a large communication step size of 0.1 (Figure 12). If the step size is reduced to 0.001 the reference values are met, c.f. Figure 13. There are further investigations necessary to identify the influence of a “nearly” high index to properties of the coupled simulation.

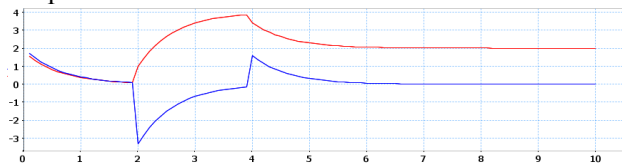


Figure 12. Result using NRM, step size 0.1

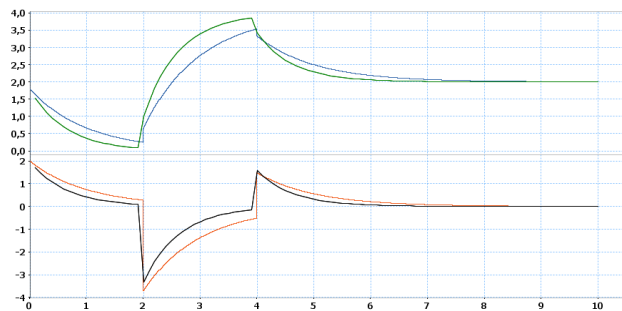


Figure 13. Comparison NRM, step sizes 0.1 and 0.001, separated into two diagrams, $x(t)$ above, $i(t)$ below.

5.5 Recommendations

The experience collected in checking test examples so far can be summarized in the following recommendations.

Essential for successful co-simulation is an intelligent tearing:

- Closely interacting parts should not be separated. Tearing is recommended at points where a signal flow direction is clearly recognized. The coupling variable should be output variable at that FMU which calculates it “significantly”. If the propagation direction of a variable changes during simulation it should not become a coupling variable.
- An output coupling variable of an FMU should have no influence to the input variables of the same FMU. At least such reactions should be delayed.
- Sometimes it is essential to transfer both the value and the derivative(s) of coupling variables. This

should be checked at each coupling variable, e.g. using test simulations.

Often these recommendations regarding tearing cannot be followed, since FMUs for components may be predetermined by specialized simulation tools. In such cases it is advantageous to revise the definition of interfaces.

Furthermore, a suitable choice of master algorithm parameters is important:

- The communication step size should be small because of numerical reasons, and large because of performance. The estimation of time constants, and test simulations can help to choose a reasonable step size. If time events (changing of discrete variables) are known they should coincide with end points of communication intervals.
- Cycles in the connection graph should be handled using a small communication step size.
- If the GSM diverges both the tearing and the definition of the direction of coupling variables should be checked.

The following points are necessary to improve the master algorithms of the test tool:

- Both the calling sequence of the FMUs and cycles in the connection graph should be defined automatically.
- It should be possible to apply different algorithms to cycles. Sometimes an algorithm should be changed during simulation.
- Variable communication step size should be introduced.
- More algorithmic parameters for the coupling methods should be introduced to adapt algorithms to given co-simulation tasks. Otherwise such adaptations should be done as automatically as possible to unburden the user.

6 Conclusion

For co-simulation of two or more FMUs three basic algorithms were described. These obvious algorithms can be a starting point for developing further coupling algorithms. Some ideas are presented.

It has to be further examined which other existing coupling algorithms can be described with the notation introduced in chapter 3, e.g. the asynchronous method (Petridis et al, 2008; Petridis, 2013). Additionally it has to be checked if other algorithms can be implemented with the existing FMI standard.

The algorithms were implemented in a master tool for testing. This master tool supports FMI 1.0 as well as FMI 2.0. It is desirable to implement further and modified algorithms which can be optimally adapted to given simulation tasks.

Many small coupling examples were developed which address special issues. These examples should be extended in future. The examples show the different behavior of the basic algorithms. The Newton-Raphson method turns out to be one of the most powerful algorithms, but its performance is usually bad. Therefore, improved algorithms should be developed.

Generally, the master tool should become more „intelligent“. It is to investigate whether internal information (the method used, the actual internal step size, numerical properties, ...) of the slave simulation within an FMU should be transferred to the master. This could help the master algorithms to be adapted better. Otherwise, a master should be capable of acting without any FMU-internal information. Both directions seem to be needed.

The presented examples are a good starting point for the FMI cross checking, because until now only single FMU are tested. With the presented examples in combination with the FMI Test Library (Otter, 2014) the coupling and with this the capability of master algorithms can be tested.

Acknowledgements

The authors are much obliged to Prof. Martin Arnold, Halle, as well as to Dr. Tom Schierz, Gilching, for any cooperation.

References

- Jens Bastian, Christoph Clauss, Susann Wolf, Peter Schneider. Master for CoSimulation Using FMI. 8th International Modelica Conference, Dresden, March 20-22, 2011.
- Christian Bertsch, Elmar Ahle, Ulrich Schulmeister. The Functional Mockup Interface – seen from an industrial perspective. 10th International Modelica Conference, March 10-12, Lund, Sweden, pp. 27-31, 2014.
- FMI project website, <https://www.fmi-standard.org/>
- Martin Otter. Modelica FMI Test Library. In: Tutorial: Functional Mockup Interface 2.0 and HiL Applications of the International Modelica Conference, Lund, Sweden, 2014
- Kosmas Petridis. Synchrone und asynchrone Verfahren zur gekoppelten Simulation mechatronischer Systeme. VDI Verlag, 2013.
- Kosmas Petridis, Andreas Klein, Michael Beitelschmidt. Asynchronous method for the coupled simulation of mechatronic systems. In: PAMM Volume 8 (2008) Nr. 1
- Tom Schierz. Modulare Zeitintegration gekoppelter Differentialgleichungssysteme in der technischen Simulation. Fortschr.-Ber. VDI Reihe 20 Nr. 447. Düsseldorf: VDI Verlag 2013.
- Tom Schierz, Martin Arnold and Christoph Clauß. Co-simulation with communication step size control in an FMI compatible master algorithm. In: Proceedings of the 9th International Modelica Conference, Munich, Germany, 2012.

Hubert Schwetlick. Numerische Lösungen nichtlinearer Gleichungen. Deutscher Verlag der Wissenschaften, Berlin, 1979, und R. Oldenbourg Verlag München, Wien, 1979.

Experimental Calibration of Heat Transfer and Thermal Losses in a Shell-and-Tube Heat Exchanger

Javier Bonilla^{1,2} Alberto de la Calle^{1,2} Margarita M. Rodríguez-García¹
Lidia Roca^{1,2} Loreto Valenzuela¹

¹CIEMAT-PSA, Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas - Plataforma Solar de Almería, Spain, {javier.bonilla, alberto.calle, margarita.rodriguez, lidia.roca, loreto.valenzuela}@psa.es

²CIESOL, Solar Energy Research Center, Joint Institute University of Almería - CIEMAT, Almería, Spain

Abstract

Many commercial solar thermal power plants rely on thermal storage systems in order to provide a stable and reliable power supply. The heat exchanger control strategies, to charge and discharge the thermal storage system, strongly affect the performance of the power plant. With the aim of developing advanced control strategies, a dynamic model of a shell-and-tube heat exchanger is being developed. This heat exchanger belongs to the CIEMAT-PSA molten salt testing facility. The goal of this facility is to study thermal storage systems in solar thermal power plants. During experimental campaigns performance losses with respect to design performance were noticed in the heat exchanger. Therefore and in order to develop an accurate heat exchanger model, thermal losses as well as heat transfer correlations on both fluid sides have been calibrated against experimental data.

Keywords: calibration, heat exchanger, heat transfer correlation, thermal losses, JModelica.org

1 Introduction

Many factors such as, environmental issues, concern about sustainability and rising cost of fossil fuels are presently encouraging research and investment into renewable resources. Renewable energy power plants face the main problem of dispatchability of demand due to the variability of their power sources. Nevertheless, solar thermal power plants are appropriate for large-scale energy production since they efficiently store heat in Thermal Energy Storage (TES) systems. Thus, many commercial solar thermal power plants rely on this technology (Herrmann and Kearney, 2002).

The performance of solar thermal power plants with TES systems is highly influenced by the heat exchanger control strategies applied in the charging and discharging processes (Zaversky et al., 2013). Therefore, advanced control strategies may improve the performance of the

whole plant. For this reason, a dynamic heat exchanger model is being developed. This heat exchanger is part of the CIEMAT-PSA molten salt testing facility. This multi-purpose molten salt testing facility is devoted to evaluate and control the heat exchange between molten salt and different kind of heat transfer fluids which could be used in solar thermal power plants.

During experimental campaigns, performance losses were noticed in the heat exchanger with respect to design performance. A dynamic heat exchanger model is being developed in order to evaluate such losses (Bonilla et al., 2015). This paper shows the followed procedure to calibrate heat exchanger thermal losses as well as heat transfer correlations for both fluid sides.

The paper is organized as follows, section 2 briefly describes the experimental plant and the heat exchanger. Section 3 carries out an analysis of heat transfer in the heat exchanger. Once this analysis is completed, heat transfer correlations in the literature are examined in section 4, thermal losses are estimated against experimental data in section 5 and heat transfer coefficients are also estimated by means of calibrating heat transfer correlations in section 6. Finally, main conclusions together with on-going work tasks are presented in section 7.

2 Experimental Plant

A multipurpose molten salt testing facility, with the goal of studying TES system, was set up at Plataforma Solar de Almería (PSA), division of CIEMAT, the public research center for Energy, Environmental and Technological Research, which is owned by the Spanish government. The CIEMAT-PSA molten salt testing facility can evaluate and control the heat exchange between molten salts and potential heat transfer fluid for solar thermal power plants, i.e. thermal oil and pressurized gases (air, CO₂, etc.). In order to use pressurized gases, this facility is connected to the innovative fluids test loop facility by means of a CO₂ - molten salt heat exchanger. This last fa-



Figure 1. CIEMAT-PSA molten salt testing facility.



Figure 2. Thermal oil - molten salt heat exchanger.

cility comprises two parabolic-trough collectors and allow studying pressurized gases as heat transfer fluids, for further information consult Rodríguez-García (2009).

The CIEMAT-PSA molten salt testing facility, shown in figure 1 is composed by hot and cold molten salt tanks, a CO₂ - molten salt heat exchanger, a thermal oil loop, two flanged pipe sections and the electrical heat tracing. The thermal oil loop comprises a thermal oil expansion tank, a centrifugal pump, an oil heater, molten salt and oil air coolers and a thermal oil - molten salt heat exchanger. This last heat exchanger is the one considered in this work, it is described in section 2.2 and it is shown in figure 2.

2.1 Operating Modes

The multipurpose molten salt testing facility can work in four different operating modes.

- *Mode 1.* Energy from the innovative fluids test loop is used to charge the molten salt TES system by means of the CO₂ - molten salt heat exchanger.
- *Mode 2.* The molten salt is cooled down by the air cooler system.
- *Mode 3.* The TES system is charged with energy coming from the thermal oil loop by means of the thermal oil - molten salt heat exchanger.
- *Mode 4.* This mode discharges the TES system by means of the thermal oil - molten salt heat exchanger and thus heating up thermal oil.

For further details about the facility and operating modes consult Rodríguez-García and Zarza (2011) and Rodríguez-García et al. (2014).

2.2 Thermal Oil Loop Heat Exchanger

The thermal oil loop heat exchanger is composed of two counter-flow multi-pass shell-and-tube units, see figure 2. The shell-side fluid is molten salt, in particular solar salt (60 % NaNO₃ and 40 % KNO₃), whereas the tube-side fluid is the commercial Therminol VP-1 thermal oil, due to its high pressure (max. 15 bar). The heat exchanger nominal operating conditions in mode 3 are shown in table 1. Each unit of the heat exchanger was designed following a Tubular Exchanger Manufacturers Association (TEMA) design, in particular a N-type front end stationary head, F-type shell and U-type rear end stationary head (NFU) design. Both units have drainage pipes at the rear end of the heat exchanger and are tilted 2° in order to facilitate their drainage. The F-type shell has two shell passes defined by a longitudinal baffle as well as two tube passes in U shape. The F-type shell is the most common and economical heat exchanger design

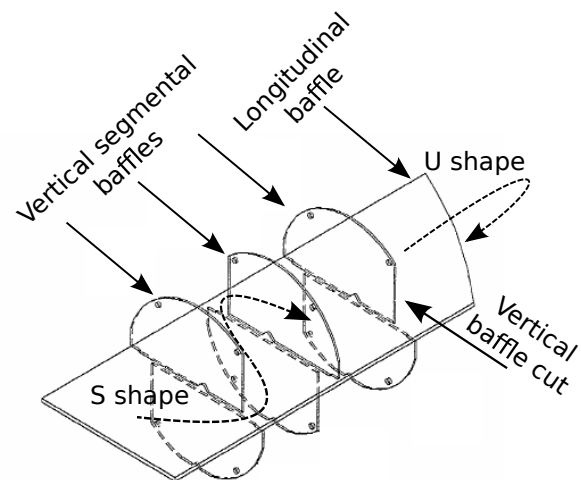


Figure 3. S-shaped and U-shaped paths along the shell side of one unit in the heat exchanger (Bonilla et al., 2015).

Table 1. Heat exchanger nominal operating conditions-mode 3

Feature	Shell side	Tube side
Fluid	Solar salt	Therminol VP-1
Inlet mass flow rate	2.08 kg/s	1.57 kg/s
Inlet pressure	2 bar	14 bar
Outlet pressure	1.6 bar	13.97 bar
Inlet temperature	290 °C	380 °C
Outlet temperature	373 °C	313 °C

used at commercial parabolic-trough solar thermal power plants (Herrmann et al., 2004). Thirty-nine vertical segmental baffles per shell pass, with vertical baffle cuts, force the shell-side fluid to follow a S-shaped path (see figure 3) in order to increase the convective heat transfer coefficient which has its highest value in cross flow. In counter flow, the tube-side fluid enters the inlet nozzle, flows along the tube bundle turning around due to the longitudinal baffle and the U-tube design, finally leaving the heat exchanger through the outlet nozzle.

3 Heat Transfer Analysis in the Heat Exchanger

Since performance losses in the heat exchanger were noticed, a heat transfer analysis considering experimental data was performed. First of all, the instrumentation installed in the facility was checked. According to the International Electrotechnical Commission (IEC) 584.3 norm, the allowable manufacturing tolerance of the K-type class 2 thermocouples is up to ± 3 °C at heat exchanger nominal operating conditions (see table 1). Nevertheless, thermocouples are periodically checked against a certified reference standard and measurements are adjusted by means of polynomials functions, therefore measurement uncertainties are reduced. Both flow meters are Yokogawa GS01F06A00-01E 50 mm volumetric vortex flow meters which have an error of up to 1 % according to the manufacturer specifications.

Secondly and due to the fact that thermocouples are not installed precisely at the inlet and outlet of the heat exchanger but rather at a certain distance, thermal losses by convection and radiation in piping along the distance between the heat exchanger and thermocouples were estimated according to eq. 1.

$$\dot{Q}_{pipe,loss} = \dot{Q}_{pipe,conv} + \dot{Q}_{pipe,rad}, \quad (1)$$

$$\dot{Q}_{pipe,conv} = h_{conv}A_{pipe}(T_{pipe} - T_{amb}), \quad (2)$$

$$\dot{Q}_{pipe,rad} = h_{rad}A_{pipe}(T_{pipe} - T_{sky}). \quad (3)$$

The piping comprises an insulated metallic tube which is protected with a thin aluminum layer. The *pipe* subscript denotes the most outer part of the pipe. Nomenclature is shown in table 2. Sky temperature (T_{sky}) is assumed to be 10 °C lower than ambient temperature. The

Table 2. Nomenclature

Latin letters			
Variable	Description	Units	
A	Area	[m ²]	
C	Heat capacity	[J/K]	
c_p	Specific heat capacity	[J/(kg K)]	
d	Diameter	[m]	
D	Characteristic dimension	[m]	
f	Friction factor	[-]	
G	Mass velocity	[kg/(m ² s)]	
h	Heat transfer coefficient	[W/(m ² K)]	
j	Chilton-Colburn j factor	[-]	
K	Thermal conductivity	[W/(m K)]	
l	Length	[m]	
m	Mass	[kg]	
\dot{m}	Mass flow rate	[kg/s]	
n	Number of measures	[-]	
Nu	Nusselt number	[-]	
Pr	Prandtl number	[-]	
\dot{Q}	Heat flow rate	[W]	
Re	Reynolds number	[-]	
t	Time	[s]	
T	Temperature	[K]	
\dot{V}	Volumetric flow rate	[m ³ /s]	
$x_1 \dots x_4$	Calibration coefficients	[-]	
y	Coefficient in ϕ	[-]	
Greek letters			
Variable	Description	Units	
ε	Emissivity	[-]	
σ	Stefan-Boltzmann constant	[W/(m ² K ⁴)]	
δ	Deviation	[%]	
ϕ	Viscosity correction factor	[-]	
ρ	Density	[kg/m ³]	
μ	Dynamic viscosity	[kg/(m s)]	
Subscript	Description	Subscript	Description
<i>amb</i>	Ambient	<i>cond</i>	Conduction
<i>conv</i>	Convection	<i>exp</i>	Experimental
<i>fluid</i>	Fluid	<i>in</i>	Inlet
<i>ins</i>	Insulation	<i>loss</i>	Losses
<i>ms</i>	Molten salt	<i>oil</i>	Thermal oil
<i>out</i>	Outlet	<i>pipe</i>	Piping
<i>rad</i>	Radiation	<i>sim</i>	Simulated
<i>sky</i>	Sky	<i>w</i>	Tube wall

heat transfer coefficient for natural convection of air over the pipe (h_{conv}) was considered 6 W/(m² K) and A_{pipe} denotes the outer surface area of the piping. The radiation heat transfer coefficient (h_{rad}) is calculated according to eq. 4, where aluminum emissivity (ε_{pipe}) was assumed to be 0.09.

$$h_{rad} = \varepsilon_{pipe} \sigma \frac{T_{pipe}^4 - T_{sky}^4}{T_{pipe} - T_{sky}}. \quad (4)$$

The outer surface piping temperature (T_{pipe}) is calculated considering that thermal losses are the same as heat con-

duction through the pipe, as it is shown in eq. 5,

$$\dot{Q}_{pipe,cond} = \dot{Q}_{pipe,conv} + \dot{Q}_{pipe,rad}, \quad (5)$$

where $\dot{Q}_{pipe,cond}$ is defined by eq. 6. It is assumed that the inner metallic tube wall temperature is the same as the fluid temperature ($T_{fluid,in}$), h_{cond} is given by eq. 7, where K_{ins} is the thermal conductivity of the insulation, l_{ins} is the insulation thickness and A_{cond} is the heat conduction area.

$$\dot{Q}_{pipe,cond} = A_{pipe}h_{cond}(T_{fluid,in} - T_{pipe}), \quad (6)$$

$$h_{cond} = \frac{K_{ins}A_{cond}}{l_{pipe}A_{pipe}}. \quad (7)$$

Therefore T_{pipe} is calculated by eq. 8,

$$T_{pipe} = \frac{h_{cond}T_{fluid,in} + h_{conv}T_{amb} + h_{rad}T_{sky}}{h_{cond} + h_{conv} + h_{rad}}. \quad (8)$$

Once thermal losses are calculated ($\dot{Q}_{pipe,loss}$), the desirable temperature, $T_{fluid,out}$ or $T_{fluid,in}$, depending on the position of the thermocouple with respect to the heat exchanger can be calculated considering eq. 9. The *in* and *out* subscripts refer to the inlet or outlet of the pipe.

$$\dot{Q}_{pipe,loss} = \dot{m}_{fluid}c_{p,fluid}(T_{fluid,out} - T_{fluid,in}). \quad (9)$$

The specific heat capacity of the fluid ($c_{p,fluid}$) can be calculated from thermodynamic properties of the particular fluid under consideration, Therminol VP-1 thermal oil (Solutia, 2008) or solar salt (Zavoico, 2001; Ferri et al., 2008) thermodynamic properties.

Once thermal losses in piping have been estimated, thermal oil (\dot{Q}_{oil}) and molten salt (\dot{Q}_{ms}) heat flow rates inside the heat exchanger should have close values in steady-state conditions, otherwise this means there are thermal losses in the heat exchanger. Heat flow rates inside the heat exchanger have been calculated considering the energy balance equation in both fluids, according to eqs. 10 and 11. The *in* and *out* subscripts refer to the heat exchanger, i.e. at the inlet or outlet of the heat exchanger.

$$\dot{Q}_{oil} = \dot{m}_{oil}c_{p,oil}(T_{oil,out} - T_{oil,in}), \quad (10)$$

$$\dot{Q}_{ms} = \dot{m}_{ms}c_{p,ms}(T_{ms,out} - T_{ms,in}). \quad (11)$$

Thermal oil and molten salt heat flow rates have been evaluated considering experimental data. The deviation between both heat flow rates has been calculated according to eq. 12.

$$\delta = 100 \frac{|\dot{Q}_{oil} - \dot{Q}_{ms}|}{\frac{1}{2}(\dot{Q}_{oil} + \dot{Q}_{ms})}. \quad (12)$$

Thermal oil and molten salt heat flow rate uncertainties inside the heat exchanger have been calculated according

Table 3. Standard uncertainties in heat flow rate variables

Var.	Standard uncertainty		Comments
	Value	Reference	
T	0.42 °C	Absolute	Periodically checked.
\dot{V}_{oil}	0.75 %	Relative	Manufacturer specs.
\dot{V}_{ms}	1.00 %	Relative	Manufacturer specs.
ρ	0.50 %	Relative	(Janz et al., 1972)
c_p	1.55 %	Relative	(Gomez et al., 2012)

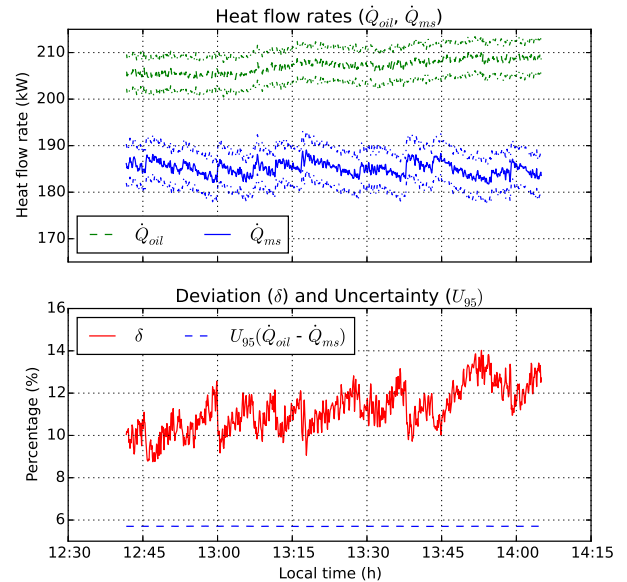


Figure 4. Steady-state case: thermal oil and molten salt heat flow rate deviation.

to the ISO/IEC Guide 98:-3:2008 Uncertainty of measurement (GUM) (International Organization of Standardization, 2008). Standard uncertainties of variables involved in eqs. 10 and 11 are given in table 3, considering volumetric flow meters for both fluids ($\dot{m} = \rho\dot{V}$). The uncertainty at a level of confidence of 95% (coverage factor $k = 2$) of the difference between thermal oil and molten salt heat flow rates, considering mode 3 nominal operating conditions, is $U_{95}(\dot{Q}_{oil} - \dot{Q}_{ms}) = 5.70\%$. Figure 4 shows both heat flow rates with their uncertainty bounds in an steady-state experiment at mode 3 nominal operating conditions. It can be seen in Figure 4 that there are thermal losses in the heat exchanger. Therefore thermal losses must be estimated in order to calculate heat transfer coefficients for this heat exchanger. Section 5 presents how thermal losses have been estimated, but before that, section 4 introduces which expressions for heat transfer correlations have been considered.

4 Heat Transfer Correlations

Experimental heat transfer correlations are commonly used in engineering calculations of heat transfer. In order to develop such heat transfer correlations, it is required to perform experiments to obtain experimental data and

also to correlate experimental data with appropriate expressions which involve dimensionless numbers. Those expressions are obtained from mass, energy and momentum conservation equations. A common expression to calculate the heat transfer coefficient in fully developed turbulent flow is the Chilton-Colburn j -analogy for mass (eq. 13) and heat (eq. 14).

$$j = \frac{f}{8}, \quad (13)$$

$$j = \frac{Nu}{RePr^{1/3}}, \quad Re \geq 10000, \quad 0.7 \leq Pr \leq 160. \quad (14)$$

Eq. 15 is derived from eqs. 13 and 14, since normally the friction factor depends on the Reynolds number, $f = f(Re)$. Therefore, the Nusselts number depends on the Reynolds and Prandtl numbers, $Nu = f(Re, Pr)$, and x_1, x_2 are commonly constant coefficients.

$$Nu = x_1 Re^{x_2} Pr^{1/3}. \quad (15)$$

With the Nusselts number, the heat transfer coefficient is calculated by eq. 16, where D is the characteristic dimension.

$$h = Nu \frac{K}{D}. \quad (16)$$

Different heat transfer correlations derive from eq. 15, such as Colburn (Çengel, 2006) and Dittus and Boelter (1930) correlations. A better accuracy for estimating the heat transfer coefficient was achieved by means of the Prandtl (1910) analogy. Petukhov (1970) improved the latest, which was modified in Gnielinski (1976) as eq. 17,

$$Nu = \frac{\frac{f}{8}(Re - 1000)Pr}{1 + 12.7\sqrt{f/8}(Pr^{2/3} - 1)} \left[1 + \left(\frac{d}{l}\right)^{2/3} \right], \quad (17)$$

$$2300 \leq Re \leq 10000, \quad 0.5 \leq Pr \leq 200.$$

Eq. 17 was derived considering fluid flow in straight ducts. Although this correlation is a good approximation for the tube side of heat exchangers, the coefficients appearing on it can be adjusted experimentally, since fluid flow path in heat exchangers is commonly complex (Taler, 2013). Eq. 18 shows Gnielinski correlation with two coefficients that could be adjusted (x_3, x_4). Such coefficients have different values in the Prandtl analogy, Petukhov, and Gnielinski correlations, therefore they are suitable coefficients to be tuned.

$$Nu = \frac{\frac{f}{8}(Re - x_3)Pr}{1 + x_4\sqrt{f/8}(Pr^{2/3} - 1)} \left[1 + \left(\frac{d}{l}\right)^{2/3} \right]. \quad (18)$$

An equivalent expression to eq. 16, and commonly used to calculate the ideal cross-flow heat transfer coefficient in the shell side of heat exchangers, is given by eq. 19,

$$h = \frac{jc_p G}{Pr^{2/3}}. \quad (19)$$

This expression is used in the Bell-Delaware method, among others. The ideal heat transfer coefficient is modified for the presence of streams by means of corrections factors, such as corrections factors for baffle cut and spacing, baffle leakage, bundle bypass flow, variable baffle spacing in the inlet and outlet sections, adverse temperature gradient buildup in laminar flow, etc. Check the Taborek implementation of the Bell-Delaware method (Thulukkanam, 2013) for further information.

The mass velocity (G) takes into account the tube bank inside the shell. The ideal Colburn j factor for the shell side is expressed as eq. 20,

$$j = x_1 Re^{x_2}, \quad (20)$$

where x_1 and x_2 are constant values within an interval of Reynolds numbers. The Reynolds number is usually calculated by eq. 21,

$$Re = \frac{GD}{\mu}. \quad (21)$$

There are other versions of eqs. 17 and 19 which incorporate the viscosity correction factor (ϕ), eq. 22, in order to take into account the viscosity gradient at the wall (μ_w) versus the viscosity at the bulk mean temperature (μ) of the fluid. The y coefficient usually depends on the ratio between viscosities (Wichterle, 1990), authors propose different values in the literature.

$$\phi = \left(\frac{\mu}{\mu_w}\right)^y. \quad (22)$$

5 Calibration of Thermal Losses

Eq. 11 has been modified in order to account for thermal losses from the shell-side fluid to the ambient and eq. 23 has been obtained.

$$\dot{Q}_{ms} = \dot{m}_{ms} c_{p,ms} (T_{ms,out} - T_{ms,in}) + \dot{Q}_{loss}. \quad (23)$$

Convective heat losses have been roughly approximated considering the shell-side (T_{ms}) and ambient (T_{amb}) temperatures in the Newton's law of cooling, as shown in eq. 24. The shell-side temperature is the arithmetic mean temperature between the inlet and outlet molten salt temperatures in the heat exchanger. A_{loss} is the outer surface area of the whole heat exchanger.

$$\dot{Q}_{loss} = h_{loss} A_{loss} (T_{ms} - T_{amb}). \quad (24)$$

The heat transfer coefficient (h_{loss}) has been defined considering eq. 19. The characteristic dimension is the inner equivalent hydraulic diameter of the shell side, x_1 and x_2 from eq. 20 have been calibrated considering experimental data. Three different experimental data sets in steady state with different flow conditions have been used in the calibration process, where $(\dot{m}_{oil}, \dot{m}_{ms}) = [(1.4 \text{ kg/s}, 2.0 \text{ kg/s}), (1.4 \text{ kg/s}, 3.2 \text{ kg/s}), (1.95 \text{ kg/s}, 2.0 \text{ kg/s})]$.

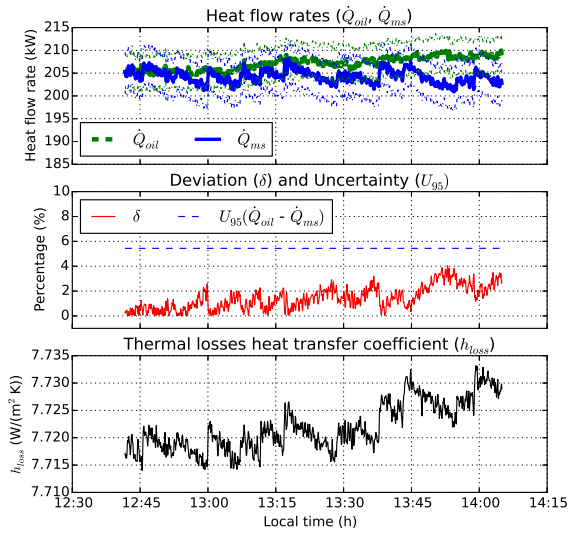


Figure 5. Steady-state case with thermal losses: thermal oil and molten salt heat flow rate deviation.

For the calibration of the x_1 and x_2 parameters, the JModelica.org tool (Åkesson et al., 2010) has been used. The optimization problem was formulated according to eq. 25, where n is the number of measures and t_i represents a time instant.

$$\min_{x_1, x_2} \sum_{i=0}^n (\dot{Q}_{oil}(t_i) - \dot{Q}_{ms}(t_i, x_1, x_2))^2. \quad (25)$$

The Nelder-Mead simplex optimization algorithm (Conn et al., 2009) performed the calibration process, the three considered experimental data sets are equally distributed, therefore each of them has $n/3$ measures. As a result of the calibration, the following values were obtained: $x_1 = 1.1858$ and $x_2 = -0.9545$. Therefore, eq. 20 is modified as eq. 26,

$$j_{loss} = 1.1858 Re_{loss}^{-0.9545}. \quad (26)$$

Heat flow rates from experimental data presented in section 3 are evaluated in figure 5, but in this case considering thermal losses according to eqs. 23, 24, 19 and 26. It can be seen that there is a good agreement between both heat flow rates since the difference is lower than the uncertainty.

Figure 6 shows heat flow rates in an experiment replicating cloud disturbances in the solar field, since the inlet thermal oil temperature was reduced and then set back to its original value. Figure 7 shows another experiment where steps in thermal oil and molten salt mass flow rates were applied. It can be seen in both figures that in steady state the deviation between both heat flow rates is lower than the uncertainty. It can be also inferred from the three experiments that the thermal losses heat transfer coefficient does not vary much and a constant value of $7.725 W/(m^2 K)$ could be assumed.

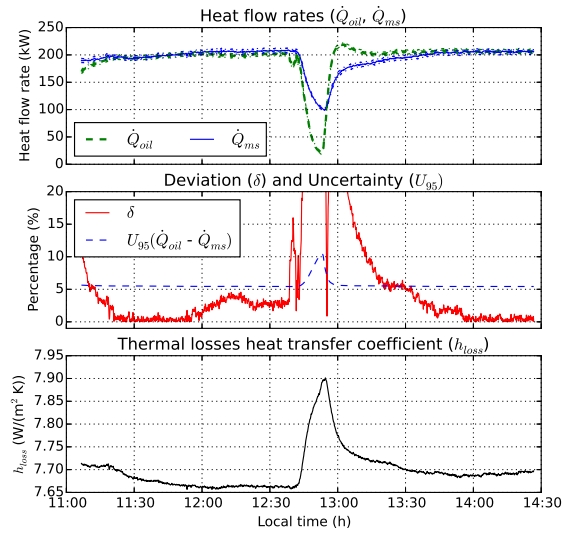


Figure 6. Cloud disturbances case: thermal oil and molten salt heat flow rate deviation.

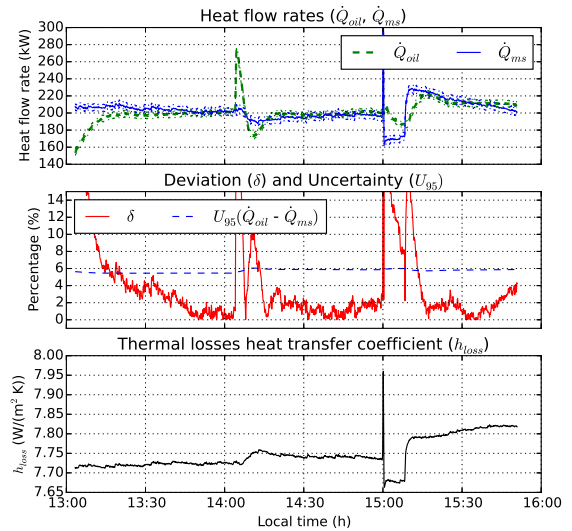


Figure 7. Mass flow rate steps case: thermal oil and molten salt heat flow rate deviation.

6 Heat Transfer Calibration

A simplified dynamic heat exchanger model has been considered in order to calibrate heat transfer correlations for the tube side as well as for the shell side. This dynamic model was presented in Correa and Marchetti (1987). It is a dynamic distributed parameter model, where each cell or Control Volume (CV) is a small lumped parameter counter-flow heat exchanger model. This model considers the thermal capacitance of the tube bundle but it neglects that of the shell metallic parts and there is neither pressure loss at the shell side nor at the tube side, thus inlet and outlet mass flow rates are equal. Eqs. 27 and 28 represent the energy balance for the tube side and the shell side respectively in each cell of the model.

$$C_{oil} \frac{dT_{oil,out}}{dt} = \dot{m}_{oil} c_{p,oil} (T_{oil,in} - T_{oil,out}) + \dot{Q}_{oil}, \quad (27)$$

$$C_{ms} \frac{dT_{ms,out}}{dt} = \dot{m}_{ms} c_{p,ms} (T_{ms,in} - T_{ms,out}) + \dot{Q}_{ms}, \quad (28)$$

where heat capacities are defined by eqs. 29 and 30,

$$C_{oil} = m_{oil} c_{p,oil} + \frac{1}{2} m_w c_{p,w}, \quad (29)$$

$$C_{ms} = m_{ms} c_{p,ms} + \frac{1}{2} m_w c_{p,w}, \quad (30)$$

and heat flow rates by eqs. 31 and 32,

$$\dot{Q}_{oil} = h A_w (T_{ms,out} - T_{oil,out}), \quad (31)$$

$$\dot{Q}_{ms} = h A_w (T_{oil,out} - T_{ms,out}) - \dot{Q}_{loss}. \quad (32)$$

The overall heat transfer coefficient (h) can be calculated by eq. 33,

$$\frac{1}{h} = \frac{1}{h_{oil}} + \frac{1}{h_{ms}}, \quad (33)$$

and thermal losses by eq. 24. Thermal losses have been already calibrated in section 5 and are included in the model by means of eqs. 24, 19 and 26.

Several heat transfer correlations have been implemented in the model and compared against experimental data. In the shell side: Gaddis and Gnielinski (VDI, 2010), the Bell-Delaware method (Thulukkanam, 2013) and a correlation proposed in Serth (2007) which is a curve fit from data provided in Kraus et al. (2002), whereas in the tube side: Gnielinski (1976), Dittus and Boelter (1930) and Hausen (1943) correlations have been also tested.

However, simulation results did not agree with experimental data. This is because there are performance losses in this heat exchanger (Bonilla et al., 2015). The most common causes for deterioration in performance of F-shell heat exchangers are thermal leakage or physical leakage due to the longitudinal baffle (Mukherjee, 2004) together with fouling, corrosion, design errors and fabrication issues. Additionally, two potential issues were identified with this heat exchanger, as presented in Rodríguez-García et al. (2014). One of them is the bypass of molten salt through the drainage channels and the other one is the nitrogen accumulation inside the shell due to the heat exchanger tilt angle. Further investigation is necessary, but in order to have an available dynamic model of the heat exchanger, heat transfer correlations have been calibrated with experimental data.

The shell-side heat transfer coefficient (h_{ms}) is defined considering eq. 19. The characteristic dimension is the outer tube diameter of the tubes in the tube bundle. The tube-side heat transfer coefficient (h_{oil}) is defined considering eq. 18, where the characteristic dimension is the inner tube diameter and the friction factor has been calculated considering the Filonenko (1954) correlation, eq. 34,

$$f_w = (1.82 \log Re_{oil} - 1.64)^{-2}. \quad (34)$$

The remaining coefficients, x_1 , x_2 (from eq. 20), x_3 and x_4 (from eq. 18) have been calibrated considering experimental data.

In Correa and Marchetti (1987), the number of cells was the number of baffles plus one multiply by the number of tube passes, however in our case that could make a total of 160 CVs, since the studied heat exchanger has 39 baffles per unit with two passes per unit. In order to reduce the time required for the calibration, the number of cells has been set to 80 CVs. Comparing simulation results, it can be stated that the maximum difference in outlet molten salt and thermal oil temperatures between the 160-CV and 80-CV models is lower than 1 °C.

The JModelica.org tool has been also used to perform the calibration process with the same experimental data sets and algorithm than for the calibration of heat losses. The optimization problem was formulated according to eq. 35.

$$\min_{x_1 \dots x_4} \sum_{i=0}^n ((T_{oil,out,exp}(t_i) - T_{oil,out,sim}(t_i, x_1, x_2))^2 + (T_{ms,out,exp}(t_i) - T_{ms,out,sim}(t_i, x_3, x_4))^2). \quad (35)$$

As a result of the calibration, the following values were obtained: $x_1 = 3.2470$, $x_2 = -1.1077$, $x_3 = 1792$ and $x_4 = 29.93$. Therefore, eqs. 20 and 18 are modified as eqs. 36 and 37,

$$j_{ms} = 3.2470 Re_{ms}^{-1.1077}, \quad (36)$$

$$Nu_{oil} = \frac{f_w (Re_{oil} - 1792) Pr_{oil}}{1 + 29.93 \sqrt{f_w} / 8 (Pr_{oil}^{2/3} - 1)} \left[1 + \left(\frac{d_w}{l_w} \right)^{2/3} \right]. \quad (37)$$

The three cases, previously analyzed in section 5, are also presented in this section in terms of temperature.

Figure 8 shows, for the steady-state case, the experimental inlet, experimental outlet and simulated outlet molten salt and thermal oil temperatures together with temperature differences between experimental and simulated outlet temperatures for both fluids. It can be seen that there is a good agreement, where the maximum difference between experimental and simulated outlet temperature for both fluid is lower than 3 °C. Figure 9 shows inlet mass flow rates and heat transfer coefficients for both fluids. Same information is shown in Figures 10 and 11, but in this case for the experiment which replicates cloud disturbances. The experimental and simulated outlet temperature differences for both fluids are lower than 5 °C in general, only when the inlet thermal oil temperature is decreased (12:40 in Figure 10), the dynamic model reacts much faster than the real system in terms of thermal oil outlet temperature. This must be further studied, it might be related to unmodeled dynamics, such as the inlet and outlet channels in the tube side of each unit in the heat exchanger, approximate heat capacities or to issues in the thermocouple. Finally, same information is also shown for the case of mass flow rate

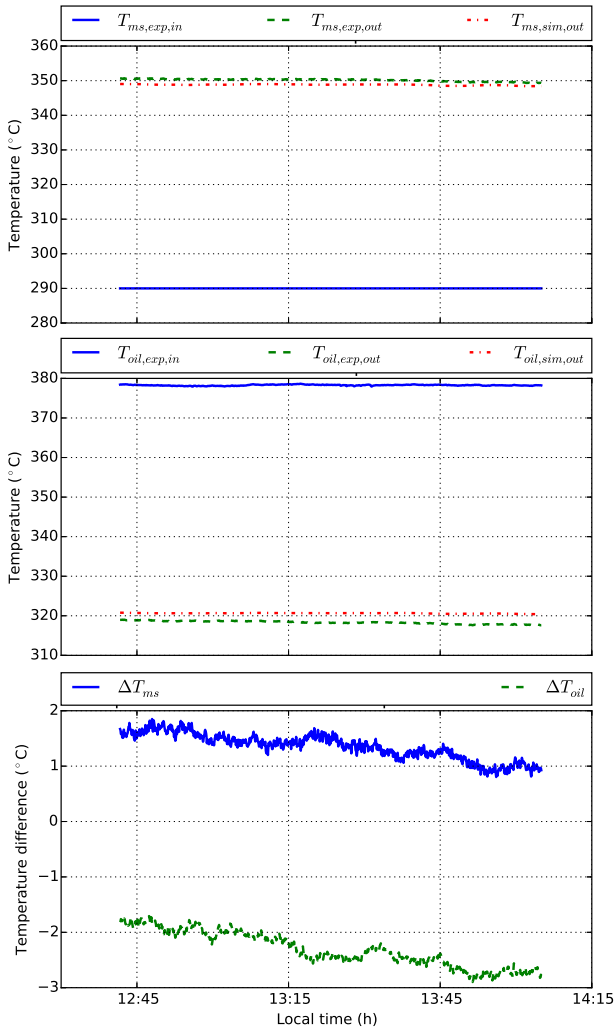


Figure 8. Steady-state case: temperatures.

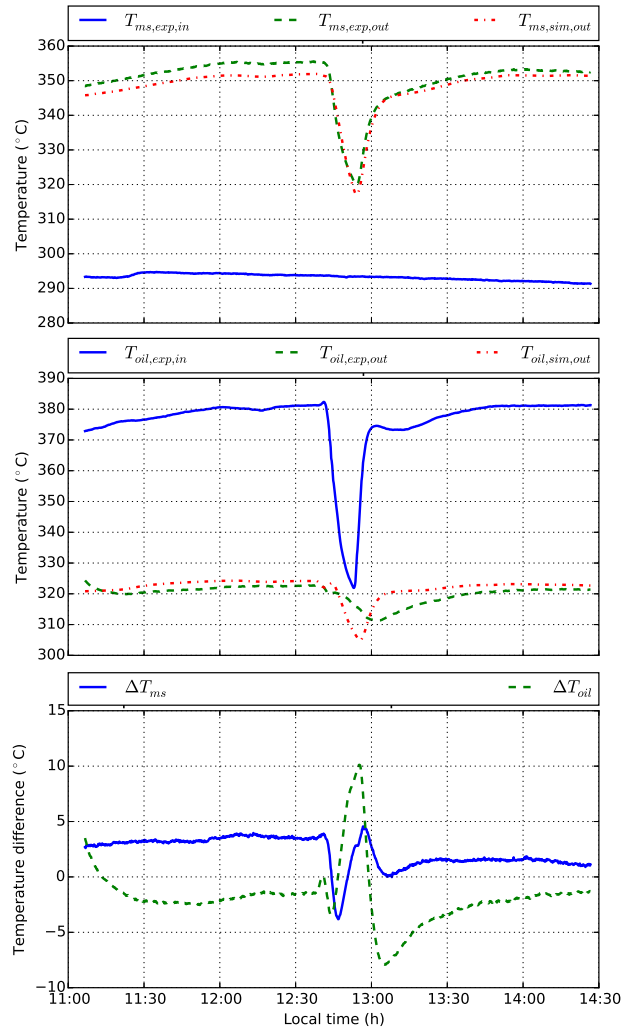


Figure 10. Cloud disturbances case: temperatures.

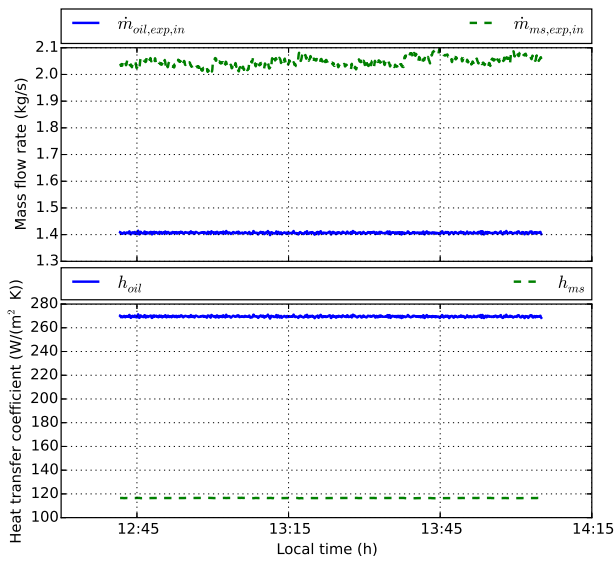


Figure 9. Steady-state case: mass flow rates and heat transfer coefficients.

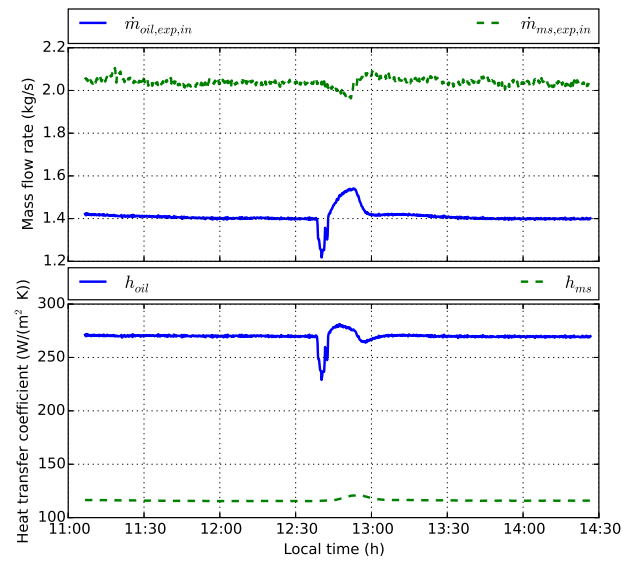


Figure 11. Cloud disturbances case: mass flow rates and heat transfer coefficients.

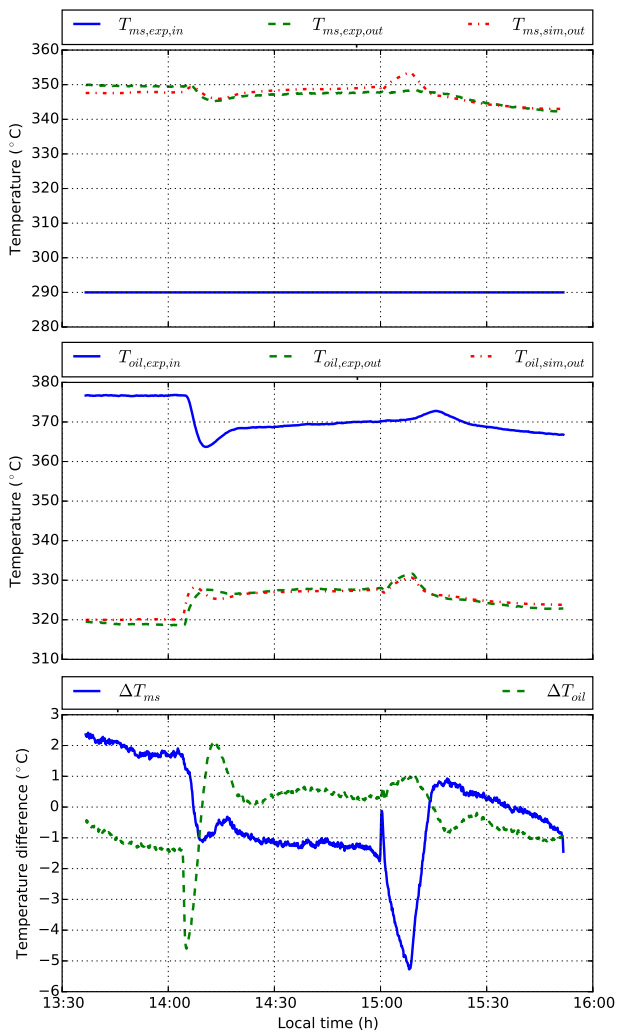


Figure 12. Mass flow rate steps case: temperatures.

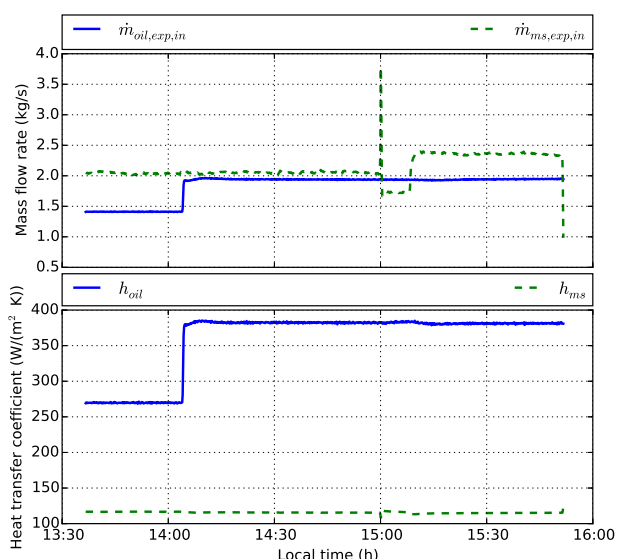


Figure 13. Mass flow rate steps case: mass flow rates and heat transfer coefficients.

steps in both fluids, as shown in Figures 12 and 13, where the experimental and simulated outlet temperature differences for both fluid are lower than 5.5 °C. There are two issues that must be studied in this case. The first one is the dynamic model response to the thermal oil mass flow rate step (14:05 in Figure 12), again the dynamic model is faster than the real system. And the second one is the increase in molten salt outlet temperature (15:00 in Figure 12), when the molten salt mass flow rate is decreased (see Figure 13). This behavior does not occur in the real system.

7 Conclusions and Ongoing Work

This paper has shown a methodology to estimate thermal losses and heat transfer correlations using Modelica and JModelica.org rather than final results, since further experimental campaigns in the facility are required in order to calibrate, if necessary, and validate the developed correlations in a wider range of operating conditions. Nevertheless, experimental data has been used to fit parameters in commonly used heat transfer correlation expressions and simulation results have been compared against experimental data with a good agreement.

Ongoing work includes integrating the calibrated correlations in a more detailed model of the heat exchanger (Bonilla et al., 2015), improving the detailed model considering a more detailed shell model as well as tube bundle model applying the cell method but particularized for a F-shell heat exchanger, as demonstrated in Zaverzky et al. (2014), studying the causes of the performance losses in the heat exchanger and performing additional experimental campaigns to validate the results.

References

- J. Åkesson, K. E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit. Modeling and optimization with Optmica and JModelica.org-Languages and tools for solving large-scale dynamic optimization problems. *Computers and Chemical Engineering*, 34(11):1737–1749, 2010. ISSN 00981354. doi:10.1016/j.compchemeng.2009.11.011.
- J. Bonilla, M.-M. Rodríguez-García, L. Roca, and L. Valenzuela. Object-Oriented Modeling of a Multi-Pass Shell-and-Tube Heat Exchanger and its Application to Performance Evaluation. In *1st Conference on Modelling, Identification and Control of Nonlinear Systems (MICNON)*, pages 107 – 112, Saint-Petersburg, Russia, 2015.
- Y. A. Çengel. *Heat Transfer: A Practical Approach (3rd edition)*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 2006. ISBN 9780072458930.
- A. Conn, K. Scheinberg, and L. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, 2009. doi:10.1137/1.9780898718768.
- D. J. Correa and J. L. Marchetti. Dynamic Simulation of Shell-and-Tube Heat Exchangers. *Heat Transfer Engineering*, 8(1):50 – 59, 1987. doi:10.1080/01457638708962787.

- F. W. Dittus and L. M. K. Boelter. Heat transfer in automobile radiators of the tubular type. *University of California Publications in Engineering*, 2(1):443–461, 1930. ISSN 07351933. doi:10.1016/0735-1933(85)90003-X.
- R. Ferri, A. Cammi, and D. Mazzei. Molten salt mixture properties in RELAP5 code for thermodynamic solar applications. *International Journal of Thermal Sciences*, 47(12):1676–1687, December 2008. ISSN 12900729. doi:10.1016/j.ijthermalsci.2008.01.007.
- G. K. Filonenko. Hydraulic drag in pipes. *Teploenergetika*, 1(4):40 – 44, 1954.
- V. Gnielinski. New equations for heat and mass transfer in turbulent pipe flow and channel flow. *International Chemical Engineering*, 2(16):359–368, 1976.
- J. C. Gomez, G. C. Glatzmaier, and M. Mehos. Heat Capacity Uncertainty Calculation for the Eutectic Mixture of Biphenyl / Diphenyl Ether Used As Heat Transfer Fluid. *SolarPaces Conference*, (September), 2012.
- H. Hausen. Darstellung des Wärmeüberganges in Rohren durch verallgemeinerte Potenzbeziehungen. *VDI - Verfahrenstechnik*, 4:91–98, 1943.
- U. Herrmann and D. W. Kearney. Survey of Thermal Energy Storage for Parabolic Trough Power Plants. *Journal of Solar Energy Engineering*, 124(2):145, 2002. ISSN 01996231. doi:10.1115/1.1467601.
- U. Herrmann, B. Kelly, and H. Price. Two-tank molten salt storage for parabolic trough solar power plants. *Energy*, 29(5-6):883–893, April 2004. ISSN 03605442. doi:10.1016/S0360-5442(03)00193-2.
- International Organization of Standardization. ISO/IEC Guide 98-3:2008 Uncertainty of measurement – Part 3: Guide to the expression of uncertainty in measurement (GUM:1995). Technical report, Switzerland, 2008.
- G. J. Janz, U. Krebs, H. F. Siegenthaler, and R. P. T. Tomkins. Molten Salts: Volume 3 Nitrates, Nitrites, and Mixtures: Electrical Conductance, Density, Viscosity, and Surface Tension Data, 1972. ISSN 00472689.
- A. D. Kraus, A. Aziz, and J. Welty. *Extended Surface Heat Transfer*. Wiley, 2002. ISBN 9780471436638.
- R. Mukherjee. Does Your Application Call for an F-Shell Heat Exchanger? *CEP magazine*, (April):40–45, 2004.
- B. S. Petukhov. Heat Transfer and Friction in Turbulent Pipe Flow with Variable Physical Properties. *Advances in Heat Transfer*, 6(C):504–564, 1970. ISSN 00652717. doi:10.1016/S0065-2717(08)70153-9.
- L. Prandtl. Eine Beziehung zwischen Wärmeaustausch und Stromungswiderstand der Flüssigkeiten. *Physik Z*, 11:1072 – 1075, 1910.
- M.-M. Rodríguez-García. First Experimental Results of a PTC Facility Using Gas as the Heat Transfer Fluid. In *15th SolarPACES Conference*, Berlin, Germany, 2009.
- M.-M. Rodríguez-García and E. Zarza. Design and Construction of an Experimental Molten Salt Test Loop. In *17th SolarPACES Conference*, Granada, Spain, 2011.
- M.-M. Rodríguez-García, M. Herrador-Moreno, and E. Zarza Moya. Lessons learnt during the design, construction and start-up phase of a molten salt testing facility. *Applied Thermal Engineering*, 62(2):520–528, January 2014. ISSN 13594311. doi:10.1016/j.applthermaleng.2013.09.040.
- R. W. Serth. *Process Heat Transfer: Principles and Applications*. Elsevier Science, 2007. ISBN 9780123735881.
- Solutia. Therminol VP-1 heat transfer fluid - Vapour and Liquid phases. Technical bulletin 7239115C, 2008.
- D. Taler. Experimental determination of correlations for average heat transfer coefficients in heat exchangers on both fluid sides. *Heat and Mass Transfer/Waerme- und Stoffuebertragung*, 49(8):1125–1139, 2013. ISSN 14321181. doi:10.1007/s00231-013-1148-5.
- K. Thulukkanam. Shell and Tube Heat Exchanger Design. In *Heat Exchanger Design Handbook, Second Edition*, Dekker Mechanical Engineering, pages 237–336. CRC Press, 2013. ISBN 978-1-4398-4212-6. doi:doi:10.1201/b14877-6.
- VDI. *VDI Heat Atlas*. Springer, 2nd edition, 2010. ISBN 9783540778769.
- K. Wichterle. A theoretical viscosity correction factor for heat transfer and friction in pipe flow. *Chemical Engineering Science*, 45(5):1343 – 1347, 1990. ISSN 00092509. doi:10.1016/0009-2509(91)85083-A.
- F. Zaversky, M. M. Rodríguez-García, J. García-Barberena, M. Sánchez, and D. Astrain. Transient behavior of an active indirect two-tank thermal energy storage system during changes in operating mode - An application of an experimentally validated numerical model. *Energy Procedia*, 49:1078–1087, 2013. ISSN 18766102. doi:10.1016/j.egypro.2014.03.117.
- F. Zaversky, M. Sánchez, and D. Astrain. Object-oriented modeling for the transient response simulation of multipass shell-and-tube heat exchangers as applied in active indirect thermal energy storage systems for concentrated solar power. *Energy*, 65:647–664, February 2014. ISSN 03605442. doi:10.1016/j.energy.2013.11.070.
- A. B. Zavoico. Solar Power Tower - Design Basis Document. Technical Report SAND2001-2100, Sandia National Laboratories, Albuquerque, USA, 2001.

Acknowledgments

This research has been funded by the EU 7th Framework Programme (Theme Energy 2012.2.5.2) under grant agreement 308912 - HYSOL project - Innovative Configuration of a Fully Renewable Hybrid CSP Plant and the Spanish Ministry of Economy and Competitiveness through ERDF and PLAN E funds (C.N. SolarNOVA ICT-CEPU 2009-02).

Suitability of Different Real-Time Solvers for a Model-Based Engineering Toolchain using Industrial Rexroth Controllers

Nils Menager¹ Rüdiger Kampfmann¹ Niklas Worschech¹ Lars Mikelsons¹

¹Bosch Rexroth AG, Lohr a. Main, Germany {nils.menager, fixed-term.ruediger.kampfmann, niklas.worschech, lars.mikelsons}@boschrexroth.de

Abstract

Due to the increasing complexity of technical systems, model-based engineering is getting more and more important during the development process of new products. The code generation from models and the usage of this code on hardware targets is one important feature of model-based development. To execute this code on the hardware device, a simulation runtime is additionally required, which offers numerical methods to solve the model equations. To use generated code on a controller, the simulation has to be executed in real-time, which is a huge requirement for the solver. In this work, a Modelica-based open source toolchain for model-based engineering with Rexroth controllers is presented, which is used for virtual commissioning of a typical hydro-mechanical system on a standard Rexroth PLC. Therefore, instead of parameterizing the controller directly on the real system, the control algorithm on the PLC is connected to the system model, which is additionally executed on the controller in parallel to the existing PLC application. Doing this, the commissioning times can be reduced significantly, as the commissioning process can already be started during the build-up of the system using a simulation model of the system. As hydro-mechanical systems are in general mathematically stiff, the choice of the solver for the system model equations is not arbitrary. In this work, five different real-time solvers, beginning with a simple explicit Euler through to more complex linearly implicit methods, are tested on a single hydraulic axis. Furthermore, typical issues like state events as well as algebraic loops are discussed in context of real-time simulation requirements.

Keywords: Real-time simulation, Modelica, Hardware-In-The-Loop, code generation, model-based engineering, real-time solver

1 Introduction

The increasing complexity of technical systems nowadays requires a change from conventional development methods towards model-based engineering. This means,

that the entire development process through to the commissioning of the system is supported by models. A consistent application of this approach reduces time and costs, for example by shorter iterations and avoiding multiple implementations. An important component of model-based engineering is code generation, meaning the generation of code out of simulation and engineering tools.

The generated code can be used for different fields of application. One important field is Rapid Control Prototyping. During the development process of a new technical system, generally, a simulation model of the plant and the controller is set up inside a simulation environment. Later, during the commissioning of the system, the control algorithm has to be implemented on the hardware controller. Here, until now, the existing model is not used any further. Instead, the controller architecture is implemented from scratch inside the development environment of the controller, which leads to some serious drawbacks. First, a re-implementation of existing code always means extra time and costs, which are not necessary. Second, as the existing code is re-implemented in another language (mostly PLC programming languages), it cannot be guaranteed, that the newly implemented controller behaves in the same way as the previously designed controller inside the simulation environment. Of course, a re-implementation of code always means a potential error source. To avoid these disadvantages, it is desirable to use the already existing model also on the hardware controller. This can be realized by generating code from the controller model.

Besides the use of controller models directly as controller on a hardware PLC, there are several other fields of application to use simulation models on industrial controller hardware. One is the usage of simulation models in parallel to the control algorithm on the controller for system diagnosis. The difference between the simulated behavior and a measurement on the real system may imply different errors inside the real system.

Furthermore, it is possible to detect even upcoming errors, which can reduce downtimes of systems and hence

reduces costs. It is imaginable to use the simulation model also for the control of the system using modern control strategies like Model Predictive Control. Here, with help of a dynamic model, which is set up during the development process anyway, the future behavior of the system is precalculated. This precalculation, in combination of a measurement of the current system behavior, is then used to determine an optimal control input to the system for the next time step.

There are already possibilities to generate code from simulation models and to run this code on real-time targets, for example using a toolchain based on MATLAB/Simulink. For Bosch Rexroth, this toolchain has some serious disadvantages. One is the fact, that the former described toolchain causes high costs due to the licence fees. Standard Rexroth customers have often no possibility to buy the software, which means that this toolchain is not accessible for them. Furthermore, the code generation module of Simulink is a black box. It is not possible to modify the code generation, for example if already existing basic functions (e.g. from an application programming interface) of the controller should directly be integrated into the generated code. Another disadvantage is the release frequency of MATLAB/Simulink (in general two new releases per year). As it remains unclear, whether there were changes inside the code generation module, all existing models have to be tested again with every new release of MATLAB/Simulink.

To avoid the disadvantages of the Simulink toolchain, an alternative toolchain based on Modelica models has been developed. This toolchain allows the user to execute Modelica models directly on Rexroth control hardware. To run the models on the controller, a simulation runtime is additionally necessary. Bosch Rexroth develops an own simulation runtime, written in C++. The simulation runtime manages the simulation, includes the numerical methods to solve the equations, is responsible for the data handling during the simulation and handles occurring events.

As the models should, for example, be used to control systems on real hardware targets, it is mandatory to run the execution of the controller model in real-time. The real-time simulation of a model is a huge requirement for the solver, as most of the common numerical methods (implicit methods like CVode and Radau) to solve the occurring equations cannot be used anymore, as they contain iterative elements, which make the execution time non-predictable. Hence, in this work, it is investigated, which numerical methods are suitable to simulate hydro-mechanical systems, which are in general mathematically stiff, under consideration of real-time requirements. Therefore, five different numerical ODE solver are compared regarding the suitability and accuracy of the solution.

1.1 Outline of this paper

This paper is structured as follows. In the second section, the toolchain for model-based engineering using Modelica models is described. The third chapter deals with numerical methods for real-time simulation. In this chapter, a short mathematical background on the methods is given. In the fourth chapter, a virtual commissioning of a commonly occurring hydro-mechanical system (single axis system) is performed on a Bosch Rexroth XM22 industrial controller. Therefore, the simulation model of the system is executed in parallel to the controller code on the PLC. This is realized using the toolchain described before. It is investigated, which of the real-time solver presented in chapter 3 can be used to simulate the system properly. The fifth section summarizes the results of the application on the test example and rates the different solver regarding their suitability for real-time simulations on industrial hardware controllers. This contribution ends with an outlook on further investigations.

2 Toolchain for model-based engineering

As already described in the introduction, one main toolchain used for model-based engineering is based on MATLAB/Simulink for the code-generation. Customers of small and medium-sized enterprises have often no possibility to use MATLAB/Simulink due to high licence fees. Furthermore, a toolchain based on a commercial tool has the disadvantage, that the models are in general encapsulated inside this tool. For an integrated, model-based engineering it is necessary to exchange models with other tools. Therefore, a tool independent description language is essential. Hence, an alternative toolchain has been developed. The requirements on this toolchain are discussed in the following section, while the realization of this toolchain is described after that.

2.1 Requirements on the toolchain

Bosch Rexroth offers both controller for industrial (e.g. Rexroth IndraControl XM22) and mobile (e.g. BODAS RC controller) applications. Hence, one requirement for the toolchain is to support both controller types without modifications on the controller. Further requirements result from the disadvantages already discussed in the introduction. The code generation should be modifiable and offer the possibility to add existing functions to the generated code. Additionally, the toolchain should not be mainly based on commercial tools, but on open standards. This is necessary to avoid external dependencies. Last but not least, the toolchain should be easy to use, so that engineers can intuitively make use of it.

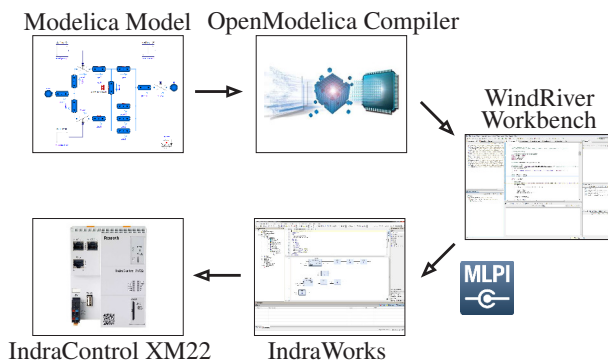


Figure 1. Structure of the Modelica-based toolchain

2.2 Realization of the toolchain

To avoid high costs and to allow an easy and uncomplicated re-use of models, the developed toolchain is based on Modelica models. Modelica is a modelling language, therefore, to generate executable code, a Modelica compiler is necessary. There exist both commercial (e.g. Dymola) and open source (e.g. OpenModelica) compiler. In this toolchain, of course, the open source OpenModelica compiler is used. One part of the compiler is the code generation. At this point, Bosch Rexroth has an own C++ code-generation. As this code generation module is self-written, it can easily be modified, for example if existing C/C++ libraries of the hardware should be used.

The generated C++-code from the code generation contains only the model. In order to execute this model, a simulation runtime is needed. The simulation runtime contains the numerical methods to solve the model equations and manages the simulation. This simulation runtime is also developed at Bosch Rexroth. Hence, it directly supports the generated code from the OpenModelica compiler. Both parts, the generated C++ code of the model and the C++ code of the simulation runtime, have then to be compiled for the control target. Therefore, a second compiler is needed. Each controller type, the industrial and the mobile controller, has its own operating system. Thus, a hardware-specific compiler is necessary.

The industrial controller used in this contribution is a Rexroth IndraControl XM22. This hardware is equipped with an Intel Atom x86 processor (1300 MHz). It works with the real-time operating system VxWorks. To generate executable code for this OS, the Windriver Workbench compiler is used. Using the Windriver Workbench compiler, both, the generated model code and the simulation runtime are compiled into a library, which is then, using the functionality of the *Motion Logic Programming Interface* (Engels and Gabler (2012)), integrated into a PLC project. The MLPI is an in-house developed interface (available in different languages as C/C++, C#, LabView, Matlab) to access controller func-

tionalties from outside. This includes for example reading/writing controller parameters and variables, starting and stopping applications, triggering tasks or executing motion commands. Additionally, MLPI can be used to link externally implemented code to a PLC function block. For setting up Rexroth industrial PLCs, *IndraWorks* as development platform is used. The function block has input and output variables, which allow the data exchange between the simulation model and the PLC program. The PLC project running on the controller may then contain different function blocks, some of them implemented in IEC 61131 code and some of them implemented in C/C++. The structure of the toolchain is shown in Figure 1.

The basis of the mobile controller is the TriCore chip. Executable code for this target can be generated using the HighTec TriCore compiler. It is necessary to use a C-API, which contains all the essential functions needed for e.g. creating tasks or apply programs to tasks. The C-API and both, the generated code from the model and the simulation runtime, are compiled into a *.hex*-file using the HighTec TriCore compiler. This file can then be flashed onto the device using the development platform *BODAS service*.

Note, that this toolchain fulfills all the requirements discussed before. Due to the usage of the open source OpenModelica compiler, the toolchain is less cost-intensive. Furthermore, it is fully compatible to the main controller types (industrial and mobile controller) available at Bosch Rexroth without any modifications on the hardware. Because of the own in-house developed code generation module and simulation runtime, needed changes, for example to use already existing external libraries, can easily be integrated. As the generated code is integrated into the existing development platforms of the controller, the engineer can keep on working in his familiar tool, e.g. in case of the industrial controller, the generated code is simply connected to a function block instead of programming the code in IEC 61131 languages. All features of the development platform, like diagnosis or visualization features, can be used in its entirety.

3 Introduction on numerical real-time solver

The main focus in the development of ODE solvers, which are suitable for industrial problems, was to reduce the average computation time, while maintaining accuracy and robustness. Therefore, the widely spread solvers like Dassel or Radau use techniques like adaptive step-size control, i.e. only using small stepsizes, when necessary, or updating the Jacobian only, when convergence fails. This yields robust as well as effective solvers.

Unfortunately, the requirements for a real-time solver are tremendously different. A solver of this kind has to guarantee that one timestep is always finished in limited time, i.e the real-time cycle. Thus always the worst case runtime has to be considered. If a small stepsize is required at a certain step, for example in order to maintain stability, this stepsize can be used everywhere, because the real-time solver has to provide enough computation time for the smallest required stepsize. With the same arguments the Jacobian, if needed in the algorithm, can be updated in every step. But there still remains a problem: the common, i.e. implicit, solvers also require solving of at least one nonlinear system. These systems require an iterative algorithm like Newton's method. Unfortunately, convergence in a certain number of iterations cannot be guaranteed.

Hence summing up all requirements, a real-time solver is a fixed step solver without nonlinear systems. Obviously, explicit Runge-Kutta methods fulfill these requirements. Thus the easiest deputy of that family, the Euler forward, is one of the most spread solvers for real-time simulation. Unfortunately, explicit methods come along with limited stability issues. This is especially a problem while dealing with stiff systems, for example hydro-mechanical problems. In section 4 it is investigated, whether they can deal with a hydraulic single axis. Therefore, the forward Euler and the classical 4th-order Runge Kutta are tested. Additionally highly stable methods for real-time simulation are needed, in order to handle stiff problems. Usually implicit Runge Kutta methods come along with good stability issues, but also with nonlinear systems (Cellier and Kofman (2006)).

Linearizing the Runge Kutta methods yields the family of Rosenbrock methods, also known as linear implicit Runge Kutta methods. They exhibit the same stability properties, while avoiding nonlinear systems. In the next subchapter a short overview of the methods used is given.

Further problems for real-time solvers are algebraic loops and events, because they can also require iterative algorithms. State events are discussed in the following sections. Nonlinear algebraic loops always require iterative algorithms like Newton's method. This means that a worst case runtime cannot be guaranteed anymore. Thus the models simulated in this contribution are free of this kind of problem.

3.1 Rosenbrock methods

For the model equation, given in state space form:

$$\frac{\partial y}{\partial t} = f(y, t) \quad y(t_0) = y_0$$

$$f: \mathbb{R}^n \times [t_0, \infty] \rightarrow \mathbb{R}^n \quad t_0 \in \mathbb{R} \quad y_0 \in \mathbb{R}^n$$

The simplest deputy of the Rosenbrock family is the linear implicit Euler, which is defined as:

$$\left(\frac{1}{h}I - \frac{\partial f}{\partial y}(t_n, y_n)\right)u = f(t_n, y_n) + \frac{\partial f}{\partial t}(t_n, y_n)$$

$$y_{n+1} = y_n + u$$

For higher order methods more stages are required. Therefore the efficient implementation of (Hairer and Wanner (2002)) is used. One step is given by:

$$\left(\frac{1}{h\gamma_{ii}}I - \frac{\partial f}{\partial y}(t_n, y_n)\right)u_i = f\left(t_n + \alpha_i h, y_n + \sum_{j=1}^{i-1} a_{ij}u_j\right)$$

$$+ \sum_{j=1}^{i-1} \frac{c_{ij}}{h}u_j + \gamma_{ih} \frac{\partial f}{\partial t}(t_n, y_n) \quad i = 1, \dots, s$$

$$y_{n+1} = y_n + \sum_{j=1}^s m_j u_j,$$

whereas $\gamma_{ij}, \alpha_i, c_{ij}, m_i$ are constants. Using special sets of constants, methods of different orders can be obtained. In this contribution, Rosenbrock methods requiring one, two and three function evaluations per step are considered. The Rosenbrock method with one function evaluation is the linear implicit Euler described before and has therefore order one. One additional function evaluation, together with the constants of ROS3P (Lang and Verwer (2001)) yields a method of order three, while a method of order four (ROS4L) can be obtained with three function evaluations and the constants of the L-stable method described in (Hairer and Wanner (2002)). All methods used here are A-stable, the latter is even L-stable. Note, that using these Rosenbrock methods, the number of necessary function evaluations is reduced by one, due to a smart choice of the constants (Hairer and Wanner (2002)).

The occurring Jacobian is numerically approximated using forward finite differences and is updated once a step. For some applications, using coloured Jacobians, the number of function evaluations can be reduced (Braun et al. (2012)).

3.2 State events

For each state event a corresponding zero function and a boolean condition variable are generated. When a certain state event occurs, the corresponding function changes its sign and the condition variable becomes true. Usually in offline simulation, iterative algorithms like the Bisection method are used to localize the zero crossing. For real-time applications this algorithms cannot be used due to their iterative components. Therefore, two simple

methods were implemented to localize the state events without iterative parts.

The first approach is probably the simplest event handling. A change of sign of the zero function is just recognized and the corresponding condition variable is set to true. No zero search algorithm is executed. Afterwards, the equations are evaluated repeatedly until a consistent status is reached. This means, that all condition variables do not change anymore. This procedure is known as event iteration and is required because the occurrence of one event might trigger another event. This approach corresponds to replacing all state events in the Modelica model with the *noEvent operator*. Hence, the accuracy of this method is strictly limited. Also two or more occurring zero crossings within one step cannot be handled. The big advantage of this approach is that there is only little additional effort and simple models with state events can be simulated.

The second approach used is based on linear interpolation. If a zero function changes its sign, the zero crossing is located with linear interpolation. Afterwards, by using linear interpolation, the states at the time of the zero crossing are computed and the corresponding condition is set to true. Then also the equations are evaluated repeatedly until a consistent status is reached as mentioned above. Because only a smaller step than the desired stepsize was executed, the simulation time is not synchronous to the controller clock anymore. Therefore the stepsize of the next step is increased. This procedure only requires one integration and interpolation per step, so that again not much additional effort is added for the event handling. As mentioned above, using the absolute time has to be avoided. In this case, the time relative to the last successful step is sufficient. For some problems, especially when the zero functions show nearly linear behaviour, this method provides quite good results. Also multiple zero crossings in one step can be handled. One big problem of this approach is that the occurrence of events in one step after another may lead to the fact, that the simulation time and the real time cannot get synchronous anymore.

The main disadvantage of this two approaches is that, without iterative algorithms, no certain accuracy can be guaranteed for the location of the zero crossings. This yields not only bad simulation results but also can cause inconsistent switching, i.e. the event iteration is not converging and has to be aborted after a certain number of steps. In this case, the simulation cannot be proceeded. Also only one event per integration step can be handled. This drawbacks have to be avoided at the model side. It is obvious that not all models can be simulated in real time due to their complexity. Therefore, the models have to be simplified for online simulation. Summing up, for applications which should run a long time, like control

algorithms, state events have to be strictly avoided. However, for applications which only run a certain time, like the plant model for the virtual commissioning presented in this contribution, state events can be tolerated, as long as the models can be simulated during the required time.

4 Virtual commissioning of a single axis system on a Rexroth PLC

In this contribution, a virtual commissioning of a hydro-mechanical system is performed, outlining the advantages of model-based engineering methods. As system, a single hydraulic axis is considered. The Modelica representation of this system is shown in Figure 2. The model contains a differential cylinder, a valve to control the volume flow, the pressure supply and the tank, and sensors to measure the piston position and the pressures in chamber A and B, respectively, of the cylinder. As already mentioned in section 3, for this contribution, the components of the system model are simplified with respect to events and hence do not contain any friction. The cylinder model does consider end stops, but the cylinder is only moved inside its feasible range anyway. As controller, a P controller with velocity feed forward and active damping, is used, which is also available inside the simulation environment. This motion controller has three inputs and one output. As input, the pressures in cylinder chambers A and B and the current piston position of the cylinder are required. The calculated output is the valve command value. Overall, there are four control parameter to choose. As desired aim of the control, a velocity profile for the cylinder piston is predefined. To avoid a re-implementation of the control algorithm on the PLC, the described toolchain is used to transfer the simulation model of the controller on the PLC. The integration into the PLC project is realized using a function block, which includes the required inputs and outputs, which are used to pass the signals of the machine to the control algorithm and get the command values from the controller.

Until now, following the standard product development process, the two phases *system design* and *commissioning* are handled independently and consecutively. Hence, the commissioning of the system, which means the testing and optimization of the controller code, which is executed in real time on industrial control hardware, can be started not until the entire system is built up and supplied with electricity. At this time, the system is already built inside the customers buildings and therefore ties up capital and space. If the controller code can be tested in parallel to the system design and before the real system is built up, the overall project time can be significantly reduced.

In order to test the controller code without having the

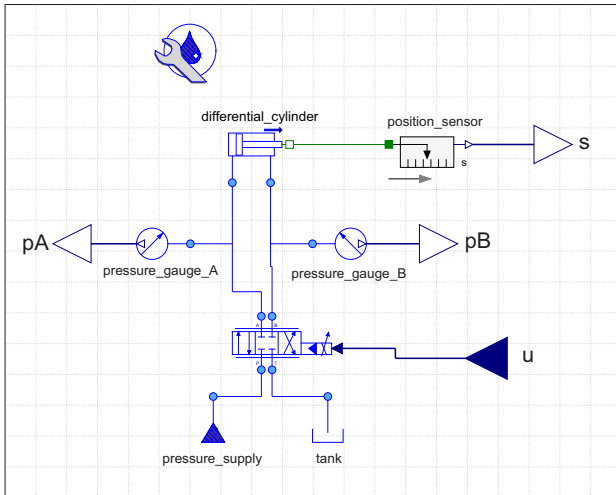


Figure 2. Simulation model of the single axis system

real system, the simulation model of the system is used to virtually commission the controller. There are several possibilities to perform the virtual commissioning. First, it is possible to run the simulation inside the simulation environment on a basic desktop computer. The hardware controller is then connected to the computer using a Hardware-In-The-Loop-setup. This approach has the disadvantage, that the simulation of the system is not executed in real time. Therefore, to synchronize the simulation and the hardware controller, the controller has to be slowed down to the simulation speed (Hofmann et al. (2015)). Hence, the real-time capability can not be verified using this strategy. A second possibility is to run the simulation on a special real-time hardware. With this approach, the real-time capability can be investigated, but additional hardware, which is only used for the commissioning, is necessary, which leads to high costs. This drawback can be avoided, if the PLC itself is used as real-time platform. As this hardware exists anyway to run the controller code, the simulation code of the plant, which is necessary for the virtual commissioning, can be executed in parallel on the same hardware. Thus, no additional hardware is needed in this case.

The simulation model of the system, which is available in Modelica, is also attached to a function block inside *IndraWorks* using the toolchain described in section 2.2. The three inputs of the controller are connected to the corresponding outputs of the function block containing the simulation model. In the same way, the output of the controller function block is connected to the input of the plant function block. To simulate the system behaviour properly, the calculation has to happen in real time. As the hydro-mechanical system is mathematically stiff, the used solver has to be chosen deliberately. The cycle time for the controller and the step size for the simulation is set to 1 ms.

4.1 Investigation on different solvers for the simulation

In a first step, before the virtual commissioning is performed, it is investigated, which solver is suitable to simulate the plant model and offers the best accuracy. Therefore, only the simulation of the system, without influences from the controller, is considered. This is necessary, because the controller can, under certain circumstances, compensate potential errors of the numerical method.

Hence, a special stimulus (sine with frequency $f = 0.5\text{Hz}$ and amplitude $\hat{y} = 2$) is applied to the input of the system, which is the input on the valve. The simulation results using the different solvers are finally compared to reference results. Reference results are obtained using the CVode solver in an offline simulation. Figure 3 shows the reference result for the output variable, the piston position of the cylinder.

Five different solver, the explicit Euler method, the explicit 4th order Runge-Kutta method as well as three different linear implicit Rosenbrock methods (order one, three and four), are used to simulate the single axis model.

4.1.1 Explicit methods

The Euler forward method is an explicit method and the easiest way to solve differential equations. Therefore, this numerical method is often used for real-time simulation. Because of the limited stability region (Cellier and Kofman (2006)), this method is not suitable for solving mathematically stiff systems like the hydro-mechanical single axis system. Using the forward Euler method in order to simulate, some variables attain physically nonsensical values (e.g. negative pressures). Hence, the simulation fails.

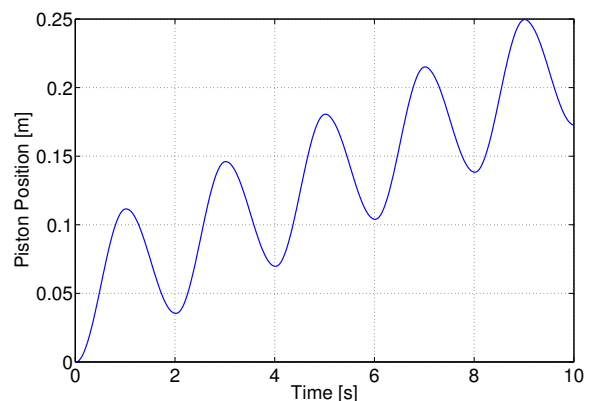


Figure 3. Reference result Piston Position generated with CVode

The explicit 4th order Runge Kutta method is also not suitable for stiff systems and shows the same behaviour as the forward Euler. Using this method, the simulation fails for the same reasons as mentioned above.

4.1.2 Rosenbrock methods

The Rosenbrock methods of order one, three and four can be used to simulate the system. In order to evaluate the accuracy of the method, the difference between the simulation result using each solver and the reference result is plotted. This difference can be seen in Figure 4. For the simulation, a step size of $h = 20$ ms is used.

4.2 Investigation on event handling using the example of a bouncing ball

The implemented event handling is tested in combination with the bouncing ball example. Both approaches described in chapter 3, with and without interpolation of the zero function, illustrate the physical effect, as soon as the ball hits the underground. But there are differences, when it comes to accuracy. The bouncing ball model is simulated for 0,75 s with a cycle time of 10ms and the ROS4L. During this time, the ball hits the ground exactly once. Figure 5 shows the simulation results.

Without any interpolation, the event is detected 39 mm below the underground (red curve). When using a linear interpolation to determine the zero crossing more precisely, the penetration of the ball can be reduced to 10^{-9} mm (blue curve). Even though the good accuracy might result from the fact, that the zero function for this example does not differ much from a straight line, it can be seen, that the linear interpolation yields convenient results, at least for some problems.

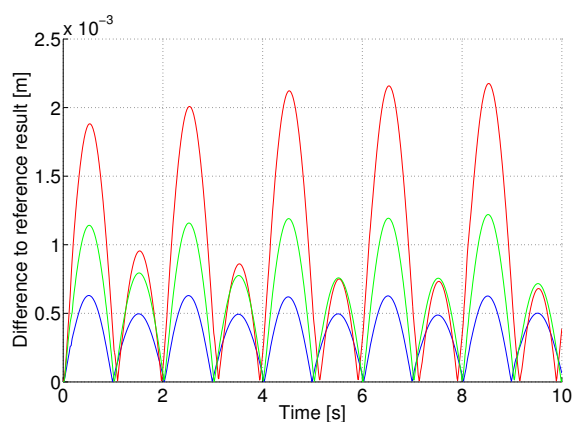


Figure 4. Difference between Rosenbrock methods and reference result [red: linear-implicit Euler; green: Rosenbrock method order 3 (ROS3P); blue: Rosenbrock method 4 (L-stable)]

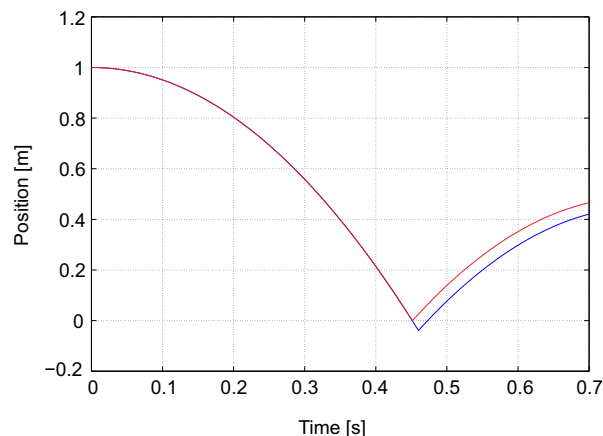


Figure 5. Bouncing Ball [red: with interpolation of zero function; blue: without interpolation of zero function]

In case, that more complex systems with multiple events are regarded, this simple state-event handling does not work. This applies for example for a complex stick-slip friction model. Hence, such effects are not included in the single axis model used in this contribution.

4.3 Practical application of the virtual commissioning

For the virtual commissioning, the stimulus described in section 4.1 is removed and the model is coupled with the controller on the PLC and then simulated in real time.

As the explicit methods fail to solve the plant model equations anyway, only the Rosenbrock methods are considered for the virtual commissioning. Within these methods, the main computational effort is not induced by the methods themselves, but through the function and Jacobian evaluations. Hence, in order to achieve the desired cycle time of 1 ms, the linear implicit Euler method is chosen for the virtual commissioning of the single axis, because it needs only one function and Jacobian evaluation per step. As shown in the passage above, this method yields the worst results, however the accuracy is still good enough for this problem.

Figure 6 shows the results of the virtual commissioning. The actual position of the piston and the desired one show a very good agreement. So the parameterization of the controller is appropriate for this problem and the controller can be used on the real system. Using the method of virtual commissioning, the commissioning time can be reduced significantly. Even though the control algorithm is tested and parameterized after the virtual commissioning, a fine adjustment of the control parameters has to be performed, as soon as the control hardware is connected to the real machine. This is necessary, be-

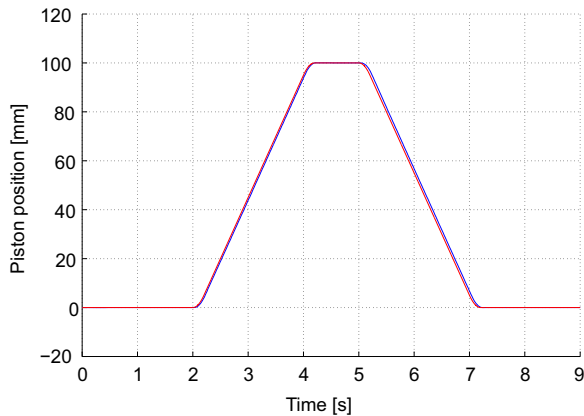


Figure 6. Comparison of the desired and actual behaviour after performing the virtual commissioning; used numerical method: Rosenbrock order 1

cause the behaviour of the virtual plant differs from the real system due to model inaccuracies.

5 Conclusion

In this work, a toolchain to run Modelica models on real hardware controller is presented. This toolchain allows the user to link both, the C/C++ code generated from the Modelica models and a simulation runtime to execute the simulation, to a PLC function block. Using this toolchain, it is possible to realize a virtual commissioning. The method of virtual commissioning allows the engineer to test and optimize the controller code before the real system exists. The controller on the PLC is therefore connected to the simulation model, which is running on the PLC, instead of the real system. The PLC acts in this case as both, a classical PLC for the control tasks and a real-time hardware target to simulate the virtual plant, at the same time. To run the simulation in real time, special numerical methods with real-time capability are necessary.

Therefore, different real-time solver were compared. In a first benchmark, the accuracy of the solver was analyzed on a mathematical stiff hydro-mechanical system (single axis system). It was shown, that explicit integration methods, such as the commonly used Euler forward and explicit Runge Kutta methods, fail to solve the occurring model equations and are therefore not usable for the simulation of plant models. The Rosenbrock methods presented in section 3.1, on the other hand, are in general suitable to solve stiff differential equations. As expected, the accuracy of the result depends on the order of the numerical method. The higher the order of the method, the smaller the error for a constant step size (see Figure 4).

When using simulation models on a hardware in real time, some issues have to be considered. While state events have to be strictly avoided, if the model, e.g. a control algorithm, is proposed to run within a real life application, they can be tolerated in models, which are only used for testing in a defined time range. This applies to the plant model used for the virtual commissioning.

6 Outlook

A different approach is, instead of designing a separate control algorithm like the P controller with velocity feed forward, to use the plant model directly for the control realizing a *Model Predictive Control*. Using this method, an optimal control problem has to be solved in every real-time cycle. The solution of the optimal control problem is equivalent to the optimal control input to the system in the next time step. The computation of the dynamic optimization problem is very time-consuming, which is a huge challenge.

Plant models, which contain numerous events, are still a problem for numerical real-time solvers. To improve the performance of the Rosenbrock methods in combination with state events is still an open task. As it is described, several state events like end stops (e.g. in the cylinder or the ground in the bouncing ball example) can be handled already now. Other events, especially deriving from friction, do not work properly today.

Especially if small cycle-times are required for the control, efficient code is essential to reduce simulation times. For the simulation of the models on the hardware target, the entire simulation runtime, which is also used for offline simulations, is utilized. This runtime contains features, like writing output files or dynamic state selection, which are not necessary for real-time simulations.

References

- W. Braun, S. Gallardo-Yances, K. Link, and B. Bachmann. Fast simulation of fluid models with colored jacobians. In *Proceedings of the 9th Modelica Conference, Munich, Germany, Modelica Association*, 2012.
- F. E. Cellier and E. Kofman. *Continuous System Simulation*. Springer, 2006.
- E. Engels and T. Gabler. Universelle Programmierschnittstelle für Motion-Logic Systeme. In *Struktur, Funktionen und Anwendung in Forschung und Lehre, Tagungsband AALE*, 2012.
- E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems*. Springer, 2002.

- A. Hofmann, S. Schweig, and L. Mikelsons. Virtuelle Inbetriebnahme mechatronischer Systeme unter Einbeziehung realer Industriesteuerungen von Bosch Rexroth. In *Tagungsband Mechatronik 2015, VDI Mechatroniktagung 2015 am 12.-13. März 2015 in Dortmund*, 2015.
- J. Lang and J. Verwer. ROS3P - an accurate third-order Rosenbrock solver designed for parabolic problems. *BIT Numerical Mathematics*, 41(4):731–738, 2001.

Integrated Engineering based on Modelica

Andreas Hofmann¹ Nils Menager¹ Issam Belhaj² Lars Mikelsons¹

¹Bosch Rexroth AG, Germany, {andreas.hofmann7, nils.menager, lars.mikelsons}@boschrexroth.de

²Dassault Systèmes, France, Issam.Belhaj@3ds.com

Abstract

The academic society claims the use of virtual engineering (i.e. simulation) since many years. Nevertheless, it is de facto rarely ever used in the automation industry. This paper presents an approach and a toolchain for an integrated, digital engineering workflow including virtual commissioning, shown at a real industrial example. In particular, a new method for virtual commissioning that allows to drop all real-time requirements is presented.

Cyber-physical production systems rising with the concepts of industry 4.0 have a complexity that conventional engineering methods cannot bear. Therefore, the time has come to finally use model-based systems engineering methodologies that were proposed years ago, e.g. (Verein Deutscher Ingenieure (2004)). Nevertheless, the automation industry acts very conservative towards new technology. This is mainly due to the distrust that model-based methods can be used in an economic manner. Within the development cycle in the automation industry CAD models are used, since they save costs compared to construction by hand. During other stages of the development cycle, virtual models are considered to be of little or no use, since the effort for modeling those images of real systems is assumed to excel the benefits. This prejudice can only be overcome by lowering the effort for modeling or increasing the value of generated models.

In this paper models generated in early development phases are re-utilized within later stages of the development cycle, like application engineering and commissioning. The re-use of models for virtual commissioning is in particular possible due to coupling of a Rexroth PLC and a (possibly non real-time) Modelica simulation using a new Modelica library. In order to obtain an development cycle that is as integrated as possible, transitions between different phases in the development cycle are tackled. First, starting with CAD data it is shown how to automatically generate a physical representation of a machinery in Modelica. Using the physical interfaces of Modelica the model can easily be extended by drive models from component manufacturers. In combination

with Bosch Rexroth PLCs, a transition towards the commissioning phase without further adaptations (e.g. complexity reduction for real-time application) is possible employing a new Modelica library. To show the entire potential of an integrated engineering workflow based on Modelica, an approach for creating control code based on a Modelica model of the control algorithm is given. By demonstrating those methods in the industrial application example of a bottling machine, it is disclosed that the assumptions of a high effort for creating simulation models, as mentioned introductory, can be disproved.

Keywords: integrated engineering, virtual commissioning, code generation, RFLP

1 Introduction

1.1 Motivation

Industry 4.0 is the central topic in automation industry. Controlled plants are replaced by highly automated, networking and self-regulating cyber-physical systems. Conventional development methods do not match for this complexity. Instead, those new rising challenges need to be faced by new product development methods.

Model-based System Engineering is something very natural. Before building a machine or more general a technical system in nearly every case a model is set up. However, in the automation industry in many cases this model is a mind model rather than a virtual model. Clearly, most of the benefits that apply for virtual models also hold for mind models, but to a lower extent. Thus the benefits of simulation (low cost experiments, available at any time, ...) are well known and highly valued, but the effort for virtual modeling is estimated higher than the saving in time.

Within a typical development cycle in the automation industry, simulation models are used for the 3D design. However, during the dimensioning of suitable components simulation is utilized only sometimes. During the control design of the complete system virtual models are hardly ever used. The reason for using models for the

design is quite obvious; using a CAD software clearly saves a lot of time compared to construction with pencil and paper. Nevertheless, a dynamic model helps to avoid over-dimensioning and thus may save costs. Last but not least usually it is not worth the effort setting up a model for control design.

Within the automation industry simulation will only earn its space if the benefits, i.e. the savings in time at last, outperforms the effort that is required for creating virtual images of the technical system. Currently, the effort for modeling is felt as very high, because every role in the development cycle models start from scratch. Models from other engineering phases are not re-used, sometimes even models from other engineering disciplines are ignored. Nevertheless, in order to successfully develop those cyber-physical systems that are rising as mentioned initially, an interaction of the different physical domains is required.

By extensive recycling of simulation models the cost for developing those virtual images can be reduced significantly. Utilizing Modelica as modeling language renders this re-use possible. Models can not only be transferred and used between different domains easily. Modelica also, since it is a describing language that needs a compiler to generate executable code anyway, can be used as basis for code generation for hardware targets like PLCs. Through its open interfaces it is also possible to include external libraries that provide features further than for dimensioning of a machinery. The models can also be re-used for virtual commissioning in coupling with real industry controls.

By integrating Modelica in the model-based system engineering methods, like the RFLP approach, a distinct improvement towards willingness to simulate of the automation industry can be made. Dassault Systèmes 3DEXPERIENCE platform (3DXP), see (Dassault Systèmes (2015)), that incorporates the RFLP approach in combination with Modelica, allows on the one hand to work with a clearly structured cross-domain development process and on the other hand renders the re-use of simulation models and the extend of model-based engineering towards the stages of application engineering and virtual commissioning possible.

1.2 Outline

In the following section a short overview about the RFLP approach is given. Although this method alone is not suitable lowering the perceived effort for creating virtual models, using Modelica as modeling language allows to extend the benefits, especially for automation industry. Section 3 focuses on the area of code generation for PLCs based on Modelica models. After a general definition of code generation is given and possible fields

of application are discussed, a toolchain is presented, which can be used to execute arbitrary C/C++ code on a Rexroth PLC. The subsequent chapter addresses the idea of virtual commissioning. Basic concepts are described and a new approach with focus on model-based system engineering is given. In section 5, the previously described technologies are combined in an example. Starting from CAD, the approach of integrated model-based engineering is shown for a bottling machine. In the last chapter the integrated engineering methodology is summarized and the paper is closed with an outlook about further development.

2 The RFLP Approach

During the process of product development, nowadays, several disciplines interact with each other. Since it is difficult to manage such concurrent multidisciplinary engineering processes it is necessary to provide products that meet the customer requirements or to create a structured development process in order to integrate all the disciplines and specialty groups into a coherent team effort. The RFLP approach, c.f. (Kleiner and Kramer (2013)) (**R**epirements engineering, **F**unctional design, **L**ogical design, **P**hysical design) as system engineering process based on the V-cycle design process, see Figure 1, permits to simplify matters by defining a system based on its fundamental aspects through essential views and their relations.

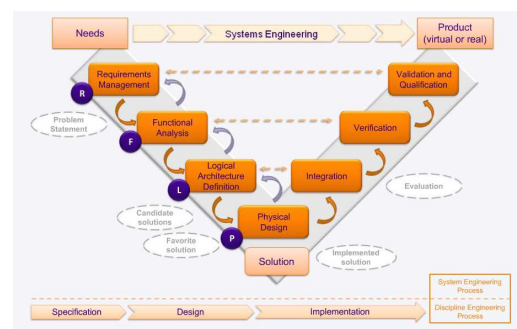


Figure 1. The steps of RFLP within the V-cycle design process.

Facilitating cross discipline communication between customers, different engineering departments, partners and suppliers, the RFLP approach provides a common view to all. RFLP ensures traceability and provides decision support in a highly collaborative environment. Furthermore, this methodology ensures that the final products meet the customers requirements in a cost effective, timely and qualitatively efficient way.

Within the **requirement engineering** customer and stakeholder needs are defined. Characteristics and activities the system has to satisfy are concentrated. Hence, the

approach permits following the life cycle of each requirement and validating each step of the engineering process. Additionally, requirements that contain physical quantities can be directly linked to those models of functional design, in which their implementation is intended.

In the **functional design** phase all capabilities of the technical system are decomposed into elementary functions. The intention of the technical system is formalized into a structure that can be allocated to technical solutions. Behavior models can be part of the functional design defining how components transform inputs to outputs. Thereby, a validation of requirements is rendered possible.

From the functional decomposition the logical architecture of the technical system is derived in the **logical design** phase. Different technological solutions corresponding to required functions are analyzed and compared during this step. Each technical solution is based on subsystems, components and their interfaces according to the technical requirements, customer needs and expected functions. Like in the functional design phase, behavior models can also be embedded into each logical component. The logical design is the main structure during the conceptional phase. Based on modular definitions, all different engineering disciplines are brought together.

Within the **physical design**, an entire virtual representation of the real technical system is modeled. The logical components are expanded by the dynamic characteristics. Mechanical models are for instance established with CAD, wires can be included, electrical drive behavior imaged or hydraulic flow represented. Eventually, the behavior of the holistic, complete cross-domain technical system can be verified against the requirements, that were defined during the requirement engineering.

2.1 3DEXPERIENCE Platform

Although RFLP can support to keep the product development cycle of a technical system clearly arranged, the limitations of virtual models that were initially introduced still remain. Furthermore, the re-use of models especially in the logical and physical phase can be difficult, if no elementary interfaces are given.

Dassault Systèmes incorporates the RFLP approach within their Business experience platform 3DXP. Since this tool provides all the different steps of product development, the interface issue between the different steps of RFLP vanishes. Furthermore, 3DXP uses Modelica for the logical and physical modeling of the technical system. This allows on the one hand to re-use the created connections from the logical model to the physical model and on the other hand an integration of different domains can be easily performed, since Modelica is

the most sophisticated modeling language for modeling complex cross-domain physical models.

2.2 Extending 3DXP towards an holistic model-based engineering

Apart from the previously stated advantages of RFLP over the product development process, the integration of Modelica offers further possibilities. As described at the beginning, effort for creating virtual models hinders many customers from automation industry to use those methods. However, this effort can be significantly reduced by expanding the use of virtual models and by re-utilizing simulation models for this enhancements. Especially in the automation industry, where the area of application engineering and commissioning are one main focus during development, simulation offers vast benefits.

3 Code generation

This section deals with code generation as a characteristic of model-based engineering. In the first part, the definition and important fields of application are discussed. After that, a toolchain to use code generation with Rexroth industrial controllers is presented.

3.1 Definition and fields of application

To realize an integrated model-based development, code generation out of simulation models is one important key feature. Code generation allows the transfer of knowledge from one development phase into following ones, e.g. between the system design and the commissioning. This technique makes it possible to re-use information, which is already available during the design phase and generally stored in a simulation model, during the commissioning. Generated code can be used for various fields of application.

The most common one is Rapid Control Prototyping. Nowadays, new technical systems are, in a first step, mostly designed virtually using modeling and simulation. Therefore, a simulation model of the system is set up inside a simulation environment. To investigate the dynamical behaviour of the plant, a control algorithm is added inside the simulation environment. After the system is completely designed and tested virtually, these models are in general not used any further. Instead, the control algorithm is re-implemented from scratch using PLC programming languages. Besides the fact, that a re-implementation needs additional time and therefore causes costs, it is always a potential error source. These disadvantages can be avoided, if the already existing controller inside the simulation model is re-used. This can

be realized by generating code out of the model and executing it on the PLC.

Besides the use of controller models on the PLC, it is also possible to use plant models on the PLC. One field of application is model-based diagnosis. The plant model is simulated in parallel to the operation of the machine. The model is used to calculate the expected dynamical behaviour of the machine, while the actual behaviour is gathered using sensors. As soon as differences between the calculated and measured values occur, this may refer to an error inside the machine. In case, that special error models are available, even the specific error type might be determined.

Code generation out of plant models can also be used for modern control strategies like Model Predictive Control. Model Predictive Control solves an optimal control problem (dynamic optimization problem) in every cycle online on the controller. The differential equations of the system, which are included in the model, are used as constraints of the optimization problem. The solution of the problem is an optimal input on the system in the next time step and minimizes a user-defined cost function, which includes the control aim.

Modelica models are perfectly suitable for code generation. As Modelica is a describing language, a compiler is needed to generate executable code. If the commercial Dymola compiler is used, C code is generated. In case of the OpenModelica compiler, both, C and C++ code, can be selected. While the Dymola code generation is a black box and can therefore not be modified, the code generation inside the OpenModelica compiler is template-based and can be easily adapted. This allows to generate specific code even for different hardware targets.

3.2 Toolchain for code generation using the OpenModelica compiler

Bosch Rexroth has an own code generation module inside the OpenModelica compiler, which generates C++ code. Using a flag, it is possible to generate code for specific industrial PLCs (e.g. IndraControl XM22). In comparison to an offline simulation, several functions of the application programming interface (API) of the controller have to be integrated into the code. The generated code contains only the model. Additional information, how the occurring model equations should be solved, is not included in the model. Therefore, a simulation runtime is necessary. This runtime includes the numerical integration methods and manages the entire simulation, e.g. handles occurring events. Bosch Rexroth develops also a runtime, written in C++, which supports the simulation of models generated from OpenModelica.

For the execution of Modelica models on industrial controllers, a toolchain is available, see (Menager et al. (2015)). This toolchain uses the OpenModelica compiler, which generates C++ code as described before. To run this code on the hardware, the code has to be compiled for the operating system on the controller. Bosch Rexroth industrial controllers use VxWorks as real-time operating system. Hence, to execute the code, a VxWorks compiler is needed. In this toolchain, the WindRiver VxWorks compiler is used to compile both, the model code and the simulation runtime, into a library. Using the Motion Logic Programming Interface (MLPI), which is used as interface to the controller, the code can be simply connected to an existing PLC application. The integration is realized with a function block, which offers the inputs and outputs for the data exchange between the executed code and the PLC. Additional information about MLPI can be found in (Engels and Gabler (2012)).

Of course, not only code generated from the OpenModelica compiler can be executed on the PLC. In general, any C or C++ code can be run on the hardware. This includes the C code, which is generated from Dymola. However, this code has to be modified manually to implement the necessary interfaces of the controller's application programming interface.

4 Virtual Commissioning

4.1 Fundamentals of Virtual Commissioning

Up to 25% of the project period of a plant and therefore a big share of the costs are needed for commissioning. Especially troubleshooting of controller application software dominates this part of the engineering process, see Figure 2.

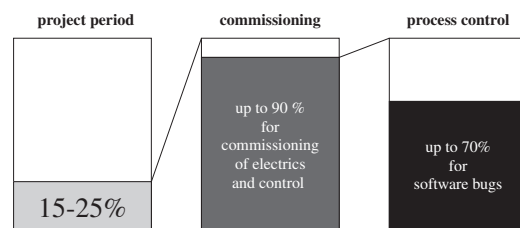


Figure 2. Amount of commissioning time on product development period, based on (VDW (1997))

Although software is a central issue already, commissioning times will most likely increase with Industry 4.0. The rise of connected, highly automated and cyber-physical systems will increase the number of software parts within a plant and thus have significant impact on commissioning times. This is, on the one hand, based on growing contents of software in mechanical and automation industry as well as horizontal and vertical networking, cf. (BMBF, 2013). On the other hand increas-

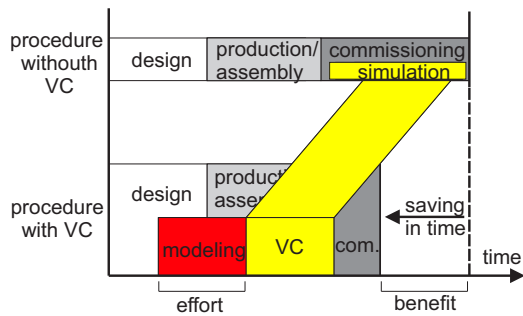


Figure 3. The basic idea of virtual commissioning, based on (Wünsch (2008))

ing complexity of software projects will also lead to rising commissioning times. The method of virtual commissioning can help reduce commissioning times significantly.

In this context virtual commissioning (VC) is understood as a method for testing functionality respectively operation sequence of a plant with the assistance of a model of the system. Basic idea of the VC is the coupling of the simulation model to a real or virtual control. Using simulation, the operability of the control application can be checked in early stages of the development. Software errors can be eliminated during production and assembly of the plant and time as well as expenses can be saved during the real commissioning, see Figure 3.

Furthermore, VC assists to increase the quality of the control application, c.f. (Wünsch (2008)). However, in order to obtain valid results the VC approach has to secure that the deterministic and consistent behavior of the PLC is represented.

4.2 Interaction between Simulation and Control

In the area of automation industry VC is typically used in the context of coupling a simulation model of the plant to a virtual or real PLC. Hence, the two approaches Hardware-in-the-Loop (HiL) and Software-in-the-Loop (SiL) are common.

Software-in-the-Loop describes the coupling of a virtual model of the plant with a virtual image of the control. Since there is no real hardware control within the setup, all models can be simulated on the same computer and no real-time requirements prevail. It follows that arbitrary complex models can be used for this kind of VC and especially models from previous engineering steps are suitable. However, this setup does not ensure deterministic execution of the control application. A full check of the system behavior is not possible.

Hardware-in-the-Loop is a VC approach in which a real PLC is coupled with a simulation model of the plant. Therefore a real-time bus and a real-time operating system is required. Of course the simulation model also has to satisfy this demands. Hence, models from other stages of the product development process are not suitable. The body of acquired knowledge in form of the simulation model needs to be discarded and a new virtual representation of the plant is required. Though, in contrast to SiL, determinism is provided, since the real control is part of the infrastructure.

In the context of an integrated engineering approach as stated preliminary, none of the prevalent methods is eligible. Either the validity of the simulation is not sufficient or simulation models from previous engineering steps can generally not be used.

4.3 Extending virtual commissioning towards MBSE

Typical coupling strategies are very limited in a simulation based development approach. In order to use VC within a model-based system engineering approach, both benefits of SiL and HiL need to be joined. For Bosch Rexroth PLCs this can be achieved using their OpenCore Interface. Bosch Rexroth OpenCore Interface, c.f. (Bosch Rexroth (2015)), is a universal port with direct access to the control and motion kernel of Rexroth industrial controls. With this technology applications can be written, using high level languages like Java or C++, that allow to join drive and control systems and conventional IT environments, see (Engels and Gabler (2012)). For the purpose of virtual commissioning the OpenCore interface is implemented in Modelica in the library `mipi4Modelica` which allows to access the control within a simulation.

Virtual commissioning with Bosch Rexroth PLCs is based on HiL abrogating the real-time requirements. This is achieved using the OpenCore technology which can interact with the motion kernel of the control. This allows to set the control in some simulation mode and tasks that are triggered by the motion cycle event, managed by the internal clock of the PLC, are no longer executed cyclically. Instead they are activated by an external trigger signal from the simulation and are launched, precisely once, during the next motion cycle. Thus the real-time requirements repeal and the complex infrastructure with real-time operating system and real-time bus can be replaced by a general simulation pc and a common ethernet connection. However, the consistent and deterministic behavior of the control is secured. Triggering the control from the simulation allows to utilize arbitrary complex models of the plant, which are in general not real-time capable.

4.4 Modelica library mlp4Modelica

The library mlp4Modelica provides functionality to directly access Bosch Rexroth PLCs and read as well as change symbol variables or parameters within the PLC program. The library itself is divided into three parts.

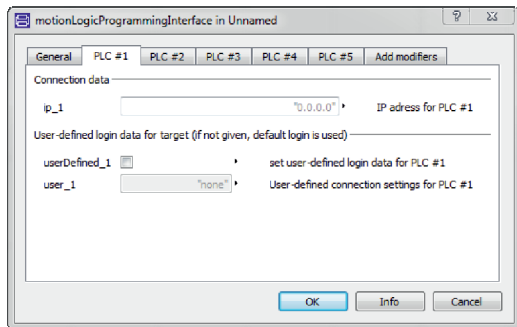


Figure 4. Parameter window of the MotionLogicProgrammingInterface model

The first part is a component called MotionLogicProgrammingInterface, which creates the connection to the PLC, c.f. Figure 4 over ethernet connection and thus is mandatory. Currently up to five industry controls can be connected. However, this limitation is due to Modelica's GUI-performance and more controls could be easily added by extending the component.

The second part of the mlp4Modelica library is represented by the model mlpCoupler and the package IO-Coupler4Modelica. The former serves as a gate, which uses the connection data from MotionLogicProgrammingInterface, and converts the signals into valid information for the PLC. It also translates the Modelica datatypes to the required datatypes on the control if possible. Furthermore, the mlpCoupler enables/disables the simulation mode of the control and regulates the trigger signal. The package IOcoupler4Modelica includes an analog and a digital interface model, that can be connected to the mlpCoupler. Within those, for every input and output, the parameter name or symbol variable name on the control respectively the PLC program is defined. Using the modelica language element connector-Sizing allows on the one hand to have not linked inputs and on the other hand to have multiple interface models connected to the mlpCoupler, c.f. Figure 5. As a consequence, arbitrary signals can be exchanged with the PLC.

The last portion of the library is the Library package, which contains all functions of the OpenCore Interface that are required for simulation purpose. Those functions, originally written in the language C, are wrapped within Modelica and allow to write own models that can access the control or the PLC application. In addition to this, when using a C/C++ code generation, this renders

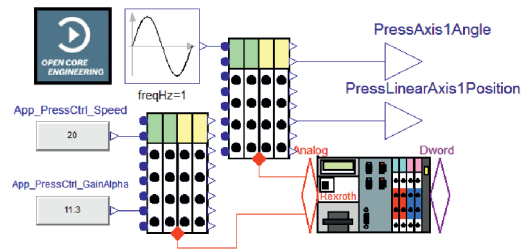


Figure 5. Example model with use of the mlpCoupler and two interface models.

creating of controller modules or even PLC application directly from the model possible as described in Section 3. It is also depicted in the following section.

5 Application Example

Bottling machines, see Figure 6, serve as good example for performing the previously described steps and extensions.

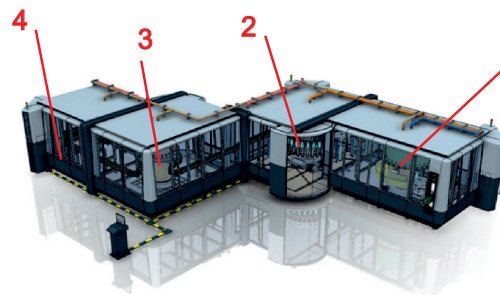


Figure 6. Image of the complete bottling machine, consisting of a rinsing machine (1), a filling machine (2), a capping system (3) and a labeling machine (4).

In this contribution the model-based system engineering approach is shown for the filling part. However, all steps are also suitable for the other machines since the methodology is, as stated before, of general purpose. The challenge in this kind of machinery is to have the bottle infeed system, the rotary filler and the bottle outfeed synchronized. Since every part has its own driving motor, the synchronization has to be performed by the PLC. However, since each motor has its own position and velocity control, the interaction has to be further investigated in order to eliminate errors due to contouring errors.

5.1 Development of the bottling machine model

The generation of dynamic models of the machinery is one big task, that hinders the use of simulation models. Within 3DXP this problem can be eliminated, at least for the mechanical part. Having the CAD data of the technical system available in 3DXP, a mechanical Modelica

model (joints, bodies, e.c.) based on the library CATIA-MultiBody can be generated, see Figure 7.

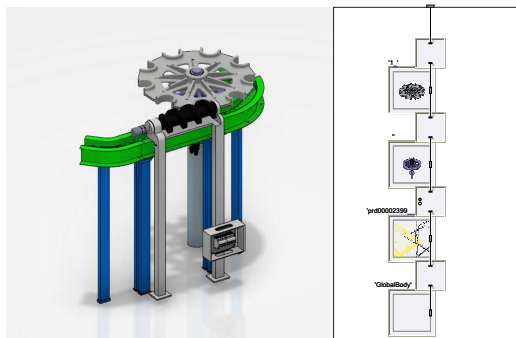


Figure 7. 3D model of the infeed system (left) and the generated Modelica model (right).

Driven joints within the 3D model are represented by joints with flanges. Therefore, a simple extension of this mechanical model is possible.

For the filling machine, as described here, the driven joints are connected to motor models from Bosch Rexroth, that contain motor specific characteristics, like torque and motor speed limitations and fine interpolation that is done by a motor controller.

The whole setup of the dynamic model of the filling machine can be seen in Figure 8.

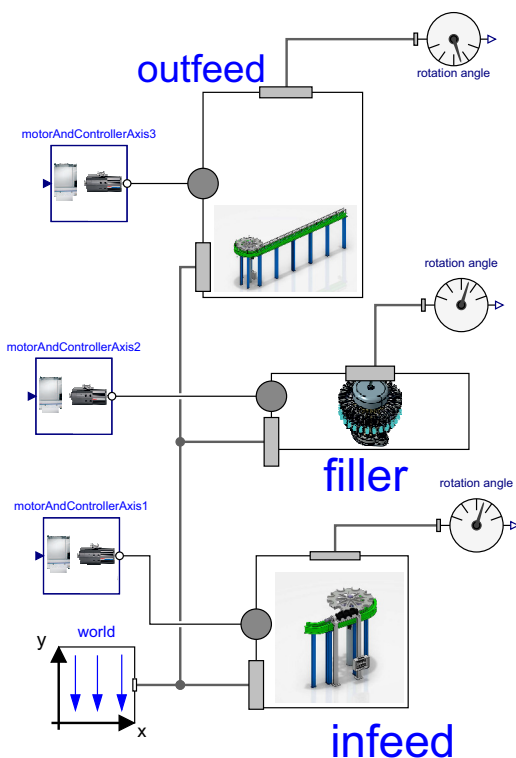


Figure 8. Dynamic model of the filling machine, divided into the mechanical model derived from CAD and the Rexroth drive models.

One can clearly see, that the motor inputs are not connected, at this point. For now, the model needs to be tested with synthetic stimuli. For the virtual commissioning of the complete system later on, it will be coupled to the real nominal axis values of the PLC and therefore receive real values.

5.2 Control design for the bottling machine

The control algorithm for the functionality of the bottling machine can be held simple. It consists basically of the states Initial, Manual, Synchronization, Automatic and Stop, see Figure 9, and was implemented in Mod- elica using the new State Machines language elements, (Modelica Association, 2014), (Elmqvist et al., 2012).

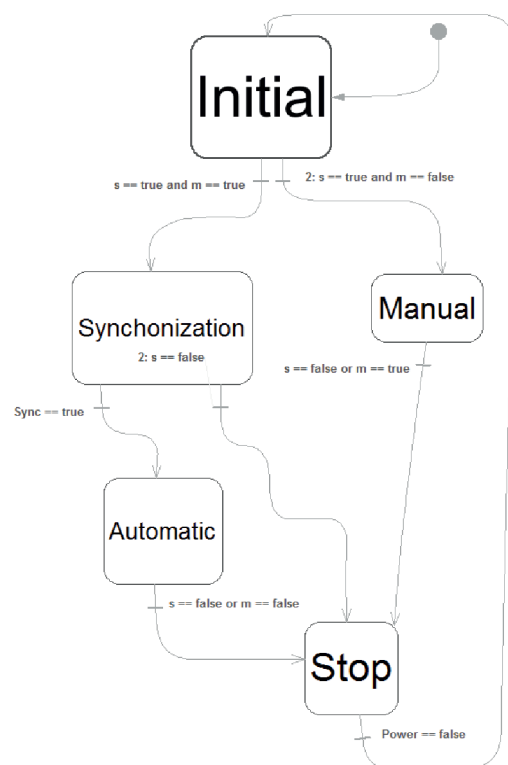


Figure 9. State machine of the bottling machine.

The transitions between the individual states are switched depending on the values of the boolean variables *start* and *automaticMode*.

State Initial

This state is active during boot up of the machinery and stays active until a transition either to State Manual ($start == true \ \&\& \ automaticMode == false$) or State Synchronization ($start == true \ \&\& \ automaticMode == true$) applies. Within this state, all three axis (infeed, filler and outfeed) of the filling machine keep in a resting position.

State Stop

Whenever the machine is running, either in state Manual, Automatic or Synchronization and a change to

any other running state or State Stop is applied, the State Stop is reached. All running axes are shut down. Switchover from State Manual (automaticMode == true || start == false) or State Synchronization (automaticMode == true || start == false) does shut down all axes individually. Transition from State Automatic (automaticMode == false || start == false) decelerates all shafts synchronously. After all axis remain in a resting position, the state automatically changes to state Initial.

State Manual

Within state Manual, all three axes of the bottle filling machine can be individually moved, depending on the values of boolean variables *axisInfeed*, *axisFiller* and *axisOutfeed* in combination with a real variable *manualSpeed*. This can be useful for manual positioning or testing of desired motor speeds.

State Synchronization and State Automatic

If the State changes from Initial to Synchronization, all Axis are moved to a synchronized position and the active state changes to State Automatic. Within the latter, all axis are accelerated in sync to a desired speed. This state represents the normal production of bottles.

All movements of axes that have been described previously are implemented using the Modelica library *mlpi4Modelica*. As a result, the complete state machine model can be used directly as a model for the real PLC. After generating C-Code from a model the compiled code can be transferred to the PLC as object program. In order to change values on the variables, e.g. *manualSpeed*, some functions block are created, see 3.

5.3 Virtual commissioning of the plant

In order to validate the behavior of the previously described controller algorithm and to examine the contouring error, that might inhibit a valid synchronization, the dynamic model is coupled to the real PLC. The interconnection is realized by the components of the library *mlpi4Modelica*, as described in section 4.

The output values from the *mlpiCoupler* are the axis nominal values that are calculated internally by the PLC. Those values are input for the internal interpolation of the drive models (motor and motor controller) from Bosch Rexroth. Their output torque drives the different mechanical parts of the filling machine. For synchronization the current angle of each axis is transferred to the PLC, see Figure 10.

After fully parametrizing the drive controller, the contouring error stays below a permitted deviation and the synchronization is not inhibited, c.f. Figure 11. Also the controller application module that was generated from the state machine model is working properly.

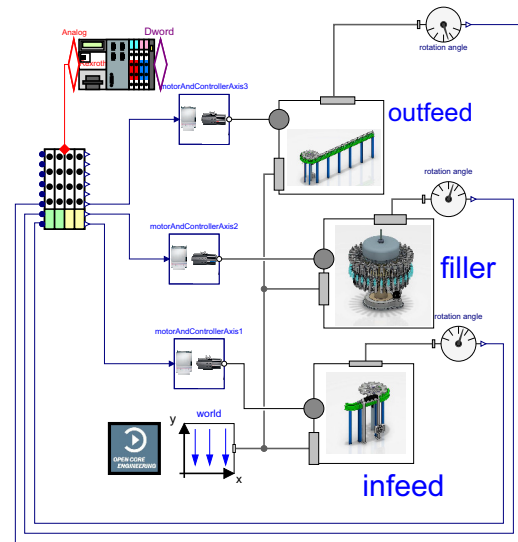


Figure 10. The complete dynamic model of the filling machine during virtual commissioning phase.

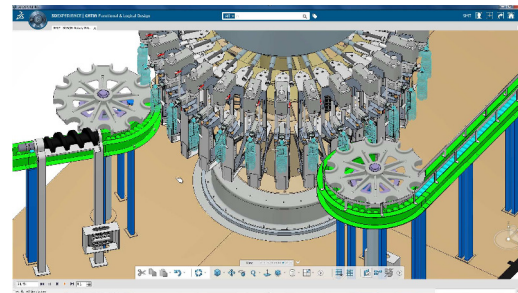


Figure 11. Filling machine in synchronized motion.

6 Conclusion and Outlook

The re-use of simulation models offers huge potentials. Utilizing those models in multiple phases of the product development cycle enhances productivity and reduces costs as well as time. Of course, the effort for creating the virtual images of a technical system remains. However, compared to the additional benefits that are available at no cost, this effort loses significance. Using Modelica as modeling language within the RFLP approach renders those enhancements of model use over the whole development cycle possible. In this contribution, code generation is used to create controller modules or whole PLC applications from simulation models. Furthermore, using the simulation model of the plant instead of the real machinery, the commissioning can be performed virtually without having produced any part yet. This is demonstrated using a bottling machine. Starting from the requirements and functional model of the plant, the steps of dynamic model generation, control code development from simulation model and virtual commissioning are described.

The benefit of simulation models does not reach the end of the line with commissioning. Since control code

can be generated from the Modelica model, it is also possible to run the simulation model on the controller during production. This enables features like model-based diagnosis or model predictive control. Also the generation of complex robot transformations which are elaborately derived by hand, can be automated using simulation models and Modelica.

References

- BMBF. *Umsetzungsempfehlung für das Zukunftsprojekt Industrie 4.0*. German Feder Ministry of Education and Research, 2013.
- Bosch Rexroth. Engineering Network: community for software developers. <http://www.boschrexroth.com/network>, 2015. Accessed 19-May-2015.
- Dassault Systèmes. 3DEXPERIENCE platform. <http://www.3ds.com/about-3ds/3dexperience-platform/>, 2015. Accessed 19-May-2015.
- Hilding Elmqvist, Fabien Gaucher, Sven Erik Mattsson, and Francois Dupont. State machines in modelica. *Proceedings of 9th International Modelica Conference*, 2012.
- Elmar Engels and Thomas Gabler. *Universelle Programmierschnittstelle für Motion-Logic Systeme : Struktur, Funktionen und Anwendung in der Forschung und Lehre*. Tagungsband AALE 2012, 2012.
- Sven Kleiner and Christoph Kramer. Model Based Design with Systems Engineering Based on RFLP Using V6. *Proceedings of the 23rd CIRP Design Conferenc*, 2013.
- Nils Menager, Lars Mikelsons, and Niklas Worschech. Model-based engineering using Rexroth controllers and open standards. *Tagungsband Mechatronik 2015*, 2015.
- Modelica Association. Modelica language specification 3.3 revision 1, 2014.
- VDW. *VDW-Bericht: Abteilungsübergreifende Projektierung komplexer Maschinen und Anlagen*. WZL, 1997.
- Verein Deutscher Ingenieure. VDI 2206: Entwicklungsmethodik für mechatronische Systeme, 2004.
- Georg Wunsch. *Methoden für virtuelle Inbetriebnahme automatisierter Produktionssysteme*. Herbert Utz Verlag, 2008.

Coupling Model Exchange FMUs for Aggregated Simulation by Open Source Tools

Pukashawar Pannu¹ Christian Andersson^{1,2} Claus Führer¹ Johan Åkesson²

¹Centre for Mathematical Sciences, Lund University, Sweden

²Modelon AB, Sweden

Abstract

The Functional Mock-up Interface standard allows to generate stand-alone sub-systems which can be simulated and verified individually. In this paper we present a design of a model aggregation which allows to simulate several Functional Mock-up Units as a coupled model. The formulation is based on Assimulo as a numerical integration environment. Assimulo problem classes are extended to a class for aggregated problems which collects information provided by the Functional Mock-up Units through the tool PyFMI together with Python based problem classes defined by Assimulo. This allows to set-up test environments of complex models composed of several sub-systems.

Keywords: FMI, Jacobian, Algebraic loops, Events, Model Exchange 2.0, Assimulo

1 Introduction

The Functional Mock-up Interface (FMI) (Blochwitz et al., 2012) has gained momentum in simulation of dynamical systems and in exchanging dynamic simulation models between tools. The standard has proven to be highly successful as it fills a gap where there were costly custom integrations before. The open source tools PyFMI¹ together with Assimulo (Andersson et al., 2015) provide a solid foundation for performing simulations and experiments on single Functional Mock-up Units (FMUs).

A key feature that is currently lacking is the ability to easily simulate coupled systems and thus fully taking advantage of the standard.

In this article, an extension to the open-source tools PyFMI and Assimulo is presented that allows for simulation of coupled model exchange FMUs following the FMI 2.0 standard. The extension enables coupling of FMUs and models written directly in Python to a so-called aggregated model.

The dynamical models considered here can be described as,

$$\dot{\bar{x}} = \bar{f}(\bar{x}, \bar{u}) \quad (1a)$$

$$\bar{y} = \bar{g}(\bar{x}, \bar{u}) \quad (1b)$$

where \bar{x} represents the states, \bar{u} the input signal and \bar{y} the output, consistent with the FMI.

Commonly, a full system model is represented by several stand-alone sub-systems coupled together by coupling equations to a model for a global system. This results in the following general system description,

$$\dot{x} = f(x, u, w) \quad (2a)$$

$$y = g(x, u, w) \quad (2b)$$

$$u = c(y, w) \quad (2c)$$

where x represents the combined states from the separate models. The local inputs for the i th model, $\bar{u}^{[i]}$, has here been separated into two vectors, $\bar{u}^{[i]} = [\hat{u}^{[i]}, \hat{w}^{[i]}]$, and subsequently combined into the global vectors (for N models), $u = [\hat{u}^{[1]}, \dots, \hat{u}^{[N]}]$ and $w = [\hat{w}^{[1]}, \dots, \hat{w}^{[N]}]$. This as to separate between inputs determined by the coupling, u , and external inputs acting on the coupled system, w . In general the external inputs can not only influence the model behaviour directly but also the coupling, Eq (2c), which is highlighted in Section 3.

When solving a coupled system, an approach is co-simulation as is explored in (Andersson, 2013) where the systems have their own integrator and the focus is on communication between systems. In this paper however, the focus is on coupling model exchange FMUs under a single solver.

2 Concept

The idea is to take N coupled sub-systems, either FMUs or Python models, and aggregate them into a single system and treating the final full system as any other model. In order to facilitate the general description of a sub-system, as is defined in FMI, for the aggregated system,

¹<http://www.pyfmi.org> PyFMI - Version 2.1. Accessed, 2015-05-18

care needs to be considered in how for example the Jacobian is defined. The Jacobian is a necessity when using implicit methods for solving the resulting system and is discussed in Section 2.3. Additionally, the events for each sub-system and external events need to be considered, discussed in Section 2.2, as well as algebraic loops which can occur due to the coupling, Section 2.4.

Now, looking at N sub-systems,

$$\dot{\bar{x}}_1^{[1]} = \bar{f}_1^{[1]}(\bar{x}_1^{[1]}, \hat{u}_1^{[1]}, \hat{w}_1^{[1]}) \quad (3a)$$

$$\bar{y}_1^{[1]} = \bar{g}_1^{[1]}(\bar{x}_1^{[1]}, \hat{u}_1^{[1]}, \hat{w}_1^{[1]}) \quad (3b)$$

$$\bar{u}_1^{[1]} = \bar{c}_1^{[1]}(\bar{y}_1^{[1]}, \hat{w}_1^{[1]}) \quad (3c)$$

⋮

$$\dot{\bar{x}}_N^{[N]} = \bar{f}_N^{[N]}(\bar{x}_N^{[N]}, \hat{u}_N^{[N]}, \hat{w}_N^{[N]}) \quad (4a)$$

$$\bar{y}_N^{[N]} = \bar{g}_N^{[N]}(\bar{x}_N^{[N]}, \hat{u}_N^{[N]}, \hat{w}_N^{[N]}) \quad (4b)$$

$$\bar{u}_N^{[N]} = \bar{c}_N^{[N]}(\bar{y}_N^{[N]}, \hat{w}_N^{[N]}) \quad (4c)$$

and the resulting aggregated system,

$$\dot{x} = f(x, u, w) = \begin{bmatrix} \bar{f}_1^{[1]}(\bar{x}_1^{[1]}, \hat{u}_1^{[1]}, \hat{w}_1^{[1]}) \\ \vdots \\ \bar{f}_N^{[N]}(\bar{x}_N^{[N]}, \hat{u}_N^{[N]}, \hat{w}_N^{[N]}) \end{bmatrix} \quad (5a)$$

$$y = g(x, u, w) = \begin{bmatrix} \bar{g}_1^{[1]}(\bar{x}_1^{[1]}, \hat{u}_1^{[1]}, \hat{w}_1^{[1]}) \\ \vdots \\ \bar{g}_N^{[N]}(\bar{x}_N^{[N]}, \hat{u}_N^{[N]}, \hat{w}_N^{[N]}) \end{bmatrix} \quad (5b)$$

$$u = c(y, w) = \begin{bmatrix} \bar{c}_1^{[1]}(\bar{y}_1^{[1]}, \hat{w}_1^{[1]}) \\ \vdots \\ \bar{c}_N^{[N]}(\bar{y}_N^{[N]}, \hat{w}_N^{[N]}) \end{bmatrix} \quad (5c)$$

The vectors x , y , u and w of the aggregated system are defined as:

$$x = \begin{bmatrix} \bar{x}_1^{[1]} \\ \vdots \\ \bar{x}_N^{[N]} \end{bmatrix}, \quad y = \begin{bmatrix} \bar{y}_1^{[1]} \\ \vdots \\ \bar{y}_N^{[N]} \end{bmatrix}, \quad u = \begin{bmatrix} \hat{u}_1^{[1]} \\ \vdots \\ \hat{u}_N^{[N]} \end{bmatrix}, \quad w = \begin{bmatrix} \hat{w}_1^{[1]} \\ \vdots \\ \hat{w}_N^{[N]} \end{bmatrix}$$

2.1 Aggregated Problem

Using the open-source tools PyFMI together with Assimulo, an FMU can be accessed from Python together with being solved using solvers available in Assimulo. With this in mind two Assimulo problem classes have been worked on. One that creates an input/output problem structure called `ExplicitProblemModel`. The other aggregates several FMUs, or `ExplicitProblemModels`,

to one large problem that can be integrated using one of Assimulo's available solvers, called `AggregatedProblem`. For simplicity an already existing problem class, `ExplicitProblem`, was extended to handle the aggregation. To define an aggregated problem class some basic data is required:

- Aggregated states.
- RHS (Right-Hand-Side) function of aggregation.
- Coupling handling.

Through PyFMI there exists already a wrapper interface that can load an FMU ME 2.0 and integrate it using Assimulo. When instantiating the `AggregatedProblem` class a list of FMUs is provided from which the initial states are easily accessible and aggregated,

for model in models:

```
    aggregated_x0 aggregate model.x0
```

The crucial part of the aggregated problem class is how to handle the right hand side function. The first major difference between an aggregated problem and an Assimulo problem is the presence of couplings. For each call to the RHS, coupling terms must be up to date. The condition can be satisfied by updating the coupling relations within the RHS-function.

Since the separate problem classes already have an RHS-function structure, computing the RHS-function of the aggregated system is simply to call the RHS-function of each sub-system,

```
set_connections()
```

for model in models:

```
    aggregated_rhs aggregate model.rhs
```

Coupling handling is done in `set_connections()`. For simple cases when for example system A input $u_2^{[A]}$ needs inputs from system B output $y_4^{[B]}$, the function simply sets $u_2^{[A]} = y_4^{[B]}$. However, this is not always the case which is further discussed in Section 2.4.

For implicit solvers a Jacobian is required and must be provided by `AggregatedProblem`. More advanced models require `AggregatedProblem` to take into account events and algebraic loops. The three mentioned topics are affected by aggregation and are discussed in the following sections.

2.2 Events

Many models include discontinuities. One way of integrating such systems is by using events (Eich-Soelner and Führer, 1998) which requires that a set of event indicators are monitored during the integration. The integration is interrupted when conditions on the event indi-

cators are violated and the event (discontinuity) is appropriately handled, finally the integration is restarted. Most of the solvers available in Assimulo have this functionality (Fredriksson et al., 2014).

The aggregated problem can access event indicators of an FMU. When asked for event indicators by an Assimulo solver the `AggregatedProblem` combines all event indicators from the sub-systems and hands them to the solver. Once an event has been detected and the integration stopped, the problem class identifies the triggered event and calls the corresponding sub-system's event handling.

Another type of events in FMUs are time events, which are known at the start of a simulation. They split up the integration into segments by setting up end times for the simulation at which point an event is handled (Andersson, 2013). This could be, for instance, a force periodically applied to the system. It is up to the aggregated system to search through all sub-systems for the closest time event to define the end time of the next integration segment and handle the event.

From the Assimulo problem design it is simple to add events to a problem. For the aggregated problem, adding of events would be to add external events to a coupled system. Events can not only be provided through the sub-systems but also through how the system is coupled. Consider a pendulum with no knowledge of its surroundings. Now, in the system model the pendulum is positioned such that its degree of freedom is limited by for instance positioning close to a wall. The limitation can be considered as an external event that needs to be taken into account. In the problem formulation this is easily done by providing extra sets of event indicators for the integrator to monitor.

2.3 Jacobian

When solving an ODE the Jacobian can be explicitly provided or numerically approximated. For an uncoupled input/output system where the inputs are only time dependent the Jacobian, $\frac{\partial f}{\partial x}$, is computed. When looking at a coupled system the dynamic changes. Due to coupling some input terms are state dependent instead of time dependent as in the uncoupled case. Consider the coupled system,

$$\dot{x} = f(x, u, w) \quad (6a)$$

$$y = g(x, u, w) \quad (6b)$$

$$u = c(y, w) \quad (6c)$$

Inserting Eq (6c) into Eq (6a) and Eq (6b) gives:

$$\dot{x} = f(x, c(y), w) \quad (7a)$$

$$y = g(x, c(y), w) \quad (7b)$$

Differentiating Eq (7a) with respect to x yields:

$$J = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial c} \frac{\partial c}{\partial y} \frac{\partial y}{\partial x} \quad (8)$$

The term $\frac{\partial y}{\partial x}$ is found by differentiating Eq (7b):

$$\frac{\partial y}{\partial x} = \frac{\partial g}{\partial x} + \frac{\partial g}{\partial c} \frac{\partial c}{\partial y} \frac{\partial y}{\partial x} \quad (9)$$

Solving for $\frac{\partial y}{\partial x}$ gives:

$$\frac{\partial y}{\partial x} = \left(I - \frac{\partial g}{\partial c} \frac{\partial c}{\partial y} \right)^{-1} \frac{\partial g}{\partial x} \quad (10)$$

Resulting in the Jacobian:

$$J = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial c} \frac{\partial c}{\partial y} \left(I - \frac{\partial g}{\partial c} \frac{\partial c}{\partial y} \right)^{-1} \frac{\partial g}{\partial x} \quad (11)$$

For the system to be solvable there is necessary condition that $(I - \frac{\partial g}{\partial c} \frac{\partial c}{\partial y})$ is non singular. The $\frac{\partial g}{\partial c}$ term handles the coupling relations and $\frac{\partial c}{\partial y}$ the sub-system feed-through terms.

With FMI 2.0 models have an option to provide directional derivatives. In case they are provided `AggregatedProblem` uses directional derivatives to approximate the aggregated Jacobian matrix. If directional derivatives are unavailable a forward difference scheme is applied. The same applies for non-FMI models.

2.4 Algebraic Loops

When a system contains feed-through, i.e. when the partial derivative of Eq (6b) with respect to u is not the zero matrix, then, in general, an equation system needs to be solved to maintain consistent input and output values satisfying,

$$y = g(x, u, w) \quad (12a)$$

$$u = c(y, w). \quad (12b)$$

By rewriting Eq (12a) to,

$$y - g(x, c(y, w), w) = 0 \quad (13)$$

the algebraic loop can be solved by an iterative method. `AggregatedProblem` creates a residual function of the left-hand-side of Eq (13) and uses the `Kinsol` solver in Assimulo to solve the problem. `Kinsol` is a non-linear algebraic equation solver, part of the SUNDIALS suite (Hindmarsh et al., 2005). When the outputs are known, Eq (12b) is used to update the inputs.

2.5 Workflow

The simulation flow of coupled systems using the aggregated problem class and an Assimulo solver is illustrated in Figure 1. The simulation flow is essentially equivalent to that of simulating an ODE with Assimulo, however, some nodes are affected by aggregation and these are coloured blue in the figure.

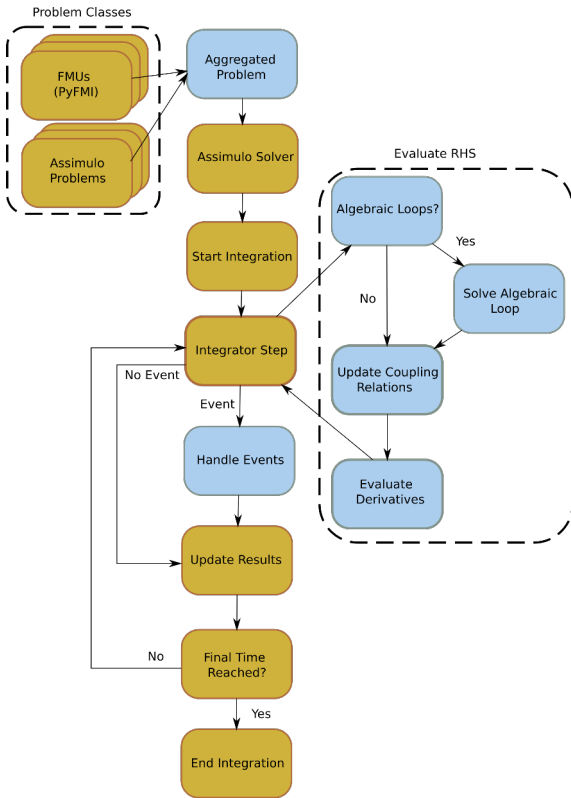


Figure 1. Assimulo simulation flow of coupled systems using FMUs, Assimulo problems and the aggregated problem class. The blue color represents nodes that are affected by aggregation.

3 Examples

In this section, the proposed framework is demonstrated. The ability to couple model exchange FMUs is shown together with coupling of FMUs with models directly defined in Python. Additionally, simulation of coupled models with externally defined events is demonstrated.

3.1 Coupled Pendula

This example demonstrates how two FMUs, each describing a pendulum, are coupled to an aggregated model. The full system consists of two pendula coupled by a force and excited by two inputs acting on the pivots.

The pendulum, with mass 1 kg and length 1 m, is described by,

$$\dot{\bar{x}}_1 = \bar{x}_3 \quad (14a)$$

$$\dot{\bar{x}}_2 = \bar{x}_4 \quad (14b)$$

$$\dot{\bar{x}}_3 = \bar{u}_1 - 2\bar{x}_1\lambda + \bar{u}_2 \quad (14c)$$

$$\dot{\bar{x}}_4 = -g - 2\bar{x}_2\lambda + \bar{u}_3 \quad (14d)$$

$$0 = \bar{x}_1^2 + \bar{x}_2^2 - 1 \quad (14e)$$

$$\bar{y}_1 = \bar{x}_1 \quad (14f)$$

$$\bar{y}_2 = \bar{x}_2 \quad (14g)$$

where \bar{x}_1, \bar{x}_2 are positions and \bar{x}_3, \bar{x}_4 velocities relative to

the pendulum's pivot. The inputs are forces, \bar{u}_2 and \bar{u}_3 , acting on the body's center and an acceleration, \bar{u}_1 due to a forced motion of the pivot. The outputs, \bar{y} , are the positions.

In order to couple two pendula, $i = [1, 2]$, the input vector is split for each pendulum into external excitations and inputs determined by the coupling,

$$\bar{u}^{[i]} = \underbrace{[\bar{u}_1^{[i]}]}_{\hat{w}^{[i]}} \underbrace{, \bar{u}_2^{[i]}, \bar{u}_3^{[i]}}_{\hat{u}^{[i]}}. \quad (15)$$

The two pendula are coupled by a linear spring which is determined by the equation, $u = c(y, w)$,

$$\begin{bmatrix} \hat{u}_2^{[1]} \\ \hat{u}_3^{[1]} \\ \hat{u}_2^{[2]} \\ \hat{u}_3^{[2]} \end{bmatrix} = k \underbrace{\begin{bmatrix} \bar{y}_1^{[1]} - a + w_1 - (\bar{y}_1^{[2]} - b - w_2) \\ \bar{y}_2^{[1]} - \bar{y}_2^{[2]} \\ -(\bar{y}_1^{[1]} - a + w_1 - (\bar{y}_1^{[2]} - b - w_2)) \\ -(\bar{y}_2^{[1]} - \bar{y}_2^{[2]}) \end{bmatrix}}_{=: \rho} \quad (16)$$

where k is the stiffness ratio. Variable a represents the pivot points x-coordinate of the left pendulum and b the point of the right pendulum. The external input vector is,

$$w = [w_1, w_2, \hat{w}^{[1]}, \hat{w}^{[2]}]. \quad (17)$$

The setup is shown in Figure 2. As previously mentioned, it is necessary to include the external inputs into the coupling as is made evident in this example. Note also, that in this example $\hat{w}^{[i]}$ has to be chosen as $\dot{w}^{[i]}$.

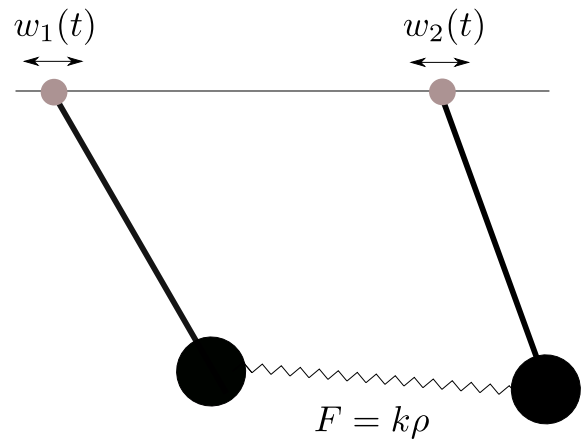


Figure 2. Two pendulums coupled via a spring.

The pendulum is modelled in the Modelica language and using the open-source tool JModelica.org (Åkesson et al., 2010) the Modelica model is compiled into an FMU. The tool is responsible for transforming the pendulum which is described as a DAE of index 3 into an ODE that FMI supports.

The aggregated system was integrated using Assimulo CVode solver with tolerances $atol = rtol = 10^{-8}$ for 5

seconds and the Jacobian approximated with forward differences. Initial conditions for the system,

$$\bar{x}_1^{[1]} = 1 \quad (18)$$

$$\bar{x}_2^{[1]} = 0 \quad (19)$$

$$\bar{x}_1^{[2]} = -1 \quad (20)$$

$$\bar{x}_2^{[2]} = 0 \quad (21)$$

note that the initial conditions are from the reference point of each pendulums pivot. The pivot points are located at $(-2,0)$ for the left pendulum and $(2,0)$ for the pendulum to the right. As external forces acting on the pivots the $\sin(t)$ function was chosen. Stiffness ratio of the spring is set to $k = 1.0$ N/m.

As reference a monolithic model of the system was created in Modelica and simulated in Dymola (Dassault Systèmes, 2016) using the solver `Dassl` with tolerance $\text{tol} = 10^{-12}$. Error of both pendulums x, y positions is presented in Figure 3 in log-scale.

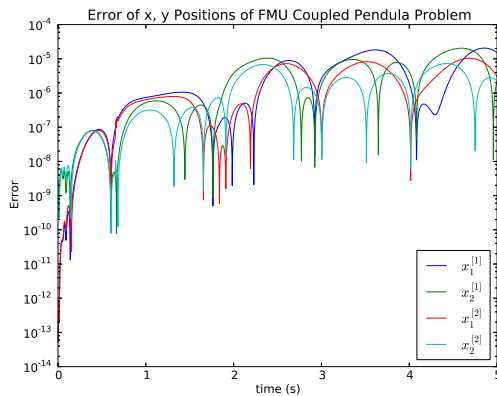


Figure 3. Error of x, y positions of aggregated coupled pendulum described in Section 3.1, simulated with CVode with $\text{atol} = \text{rtol} = 10^{-8}$ for 5 seconds. $x_1^{[i]}$ denotes the x -coordinate and $x_2^{[i]}$ the y of model i . Model [1] is the pendulum to the left and [2] the one to the right.

3.2 Coupled Pendula with Different Model Types

Three aggregated systems of coupled pendulas were modelled and compared to a monolithic reference model. The first system built by two FMU models modelled in Modelica and compiled with JModelica.org. The second by two Assimulo models and the third with the left pendulum as an Assimulo model and the right pendulum as an FMU. For this example the pendulums were modelled as ODEs in polar coordinates with unit mass and length,

$$\dot{\bar{x}}_1 = \bar{x}_2 \quad (22a)$$

$$\dot{\bar{x}}_2 = (-g + \bar{u}_3)\sin(\bar{x}_1) + (\bar{u}_2 + \bar{u}_1)\cos(\bar{x}_1) \quad (22b)$$

$$\dot{\bar{y}}_1 = \bar{x}_1 \quad (22c)$$

where g is gravitational acceleration, \bar{x}_1 is angular displacement with respect to the pivot point, \bar{x}_2 angular velocity. The inputs \bar{u}_2 and \bar{u}_3 are forces acting on the bob horizontally and vertically respectively. \bar{u}_1 is an input of acceleration due to a forced motion of the pivot. The output, \bar{y}_1 , is the angular displacement.

Similarly to the example described in Section 3.1 the input vector is split into external excitations and inputs by coupling.

$$\bar{u}^{[i]} = \underbrace{[\bar{u}_1^{[i]}]}_{\hat{w}^{[i]}} , \underbrace{[\bar{u}_2^{[i]}, \bar{u}_3^{[i]}]}_{\hat{u}^{[i]}} \quad (23)$$

The linear spring coupling the two pendulas is determined by,

$$\begin{bmatrix} \hat{u}_2^{[1]} \\ \hat{u}_3^{[1]} \\ \hat{u}_2^{[2]} \\ \hat{u}_3^{[2]} \end{bmatrix} = k \underbrace{\begin{bmatrix} (\sin(\bar{y}_1^{[1]}) - a + w_1) - (\sin(\bar{y}_1^{[2]}) - b - w_2) \\ (-\cos(\bar{y}_1^{[1]}) - (-\cos(\bar{y}_1^{[2]}))) \\ -((\sin(\bar{y}_1^{[1]}) - a - w_1) - (\sin(\bar{y}_1^{[2]}) - b - w_2)) \\ -((- \cos(\bar{y}_1^{[1]}) - (-\cos(\bar{y}_1^{[2]}))) \end{bmatrix}}_{=:p} \quad (24)$$

where k is the stiffness ratio. Variables a and b represent the pivot points x -coordinate for the left-hand-side and right-hand-side pendulas respectively. The external input vector is,

$$u_1 = [w_1, w_2, \hat{w}^{[1]}, \hat{w}^{[2]}] \quad (25)$$

As with example in Section 3.1, $\hat{w}^{[i]}$ has to be chosen as $\hat{w}^{[i]}$.

Initial conditions for the aggregated system were chosen for the pendulas to mirror each other with angles $\frac{\pi}{2}$ and $-\frac{\pi}{2}$ for the left and right pendulas and zero initial angular velocity.

$$\bar{x}_1^{[1]} = \frac{\pi}{2} \quad (26a)$$

$$\bar{x}_2^{[1]} = 0 \quad (26b)$$

$$\bar{x}_1^{[2]} = -\frac{\pi}{2} \quad (26c)$$

$$\bar{x}_2^{[2]} = 0 \quad (26d)$$

As external force exciting the pendula pivots a $\sin(t)$ signal was chosen and the springs stiffness ratio $k = 1.0$ N/m.

The aggregated system was integrated using the CVode solver in the Assimulo package with tolerances, $\text{atol} = \text{rtol} = 10^{-8}$ for a time of 5 seconds and the Jacobian approximated using forward differences. Results were then compared to a reference where the coupled pendulas were modelled as a monolithic system in Modelica and simulated with `Dassl` in Dymola using tolerance $\text{tol} = 10^{-12}$. Figure 4 shows the error in log-scale of the angle $\bar{x}_1^{[1]}$ of all three systems compared to the control. The same plot for angle $\bar{x}_1^{[2]}$ is shown in Figure 5.

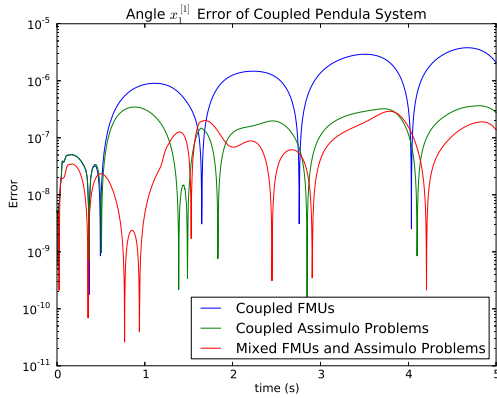


Figure 4. Error of angle $x_1^{[1]}$ of aggregated FMU, Assimulo and mixed systems, simulated for 5 seconds with tolerances $atol = rtol = 10^{-8}$ with CVode solver in Assimulo package.

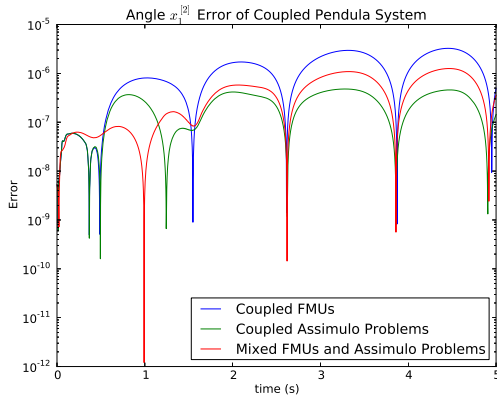


Figure 5. Error of angle $x_1^{[2]}$ of aggregated FMU, Assimulo and mixed systems, simulated for 5 seconds with tolerances $atol = rtol = 10^{-8}$ with CVode solver in Assimulo package.

3.3 Coupled Pendula Impact on Wall

The aggregated system constructed with two FMUs in Section 3.1 is here reused with the addition of discontinuities as two walls are externally placed in the path of the two pendulums' swinging motion. One wall for each pendulum. This is to highlight the possibility of externally adding state events to the coupled problem. Also the external forces acting on the pivots have been removed.

When the bob hits the wall a discontinuity occurs. This is handled by defining event indicators that trigger an event when the impact occurs. Event indicators are defined as zero-crossings as,

$$event^{[i]} = wall^{[i]} - \bar{x}_1^{[i]} \quad (27)$$

where $wall^{[i]}$ is the x-coordinate of the wall blocking pendulum $[i]$. The impact itself is elastic and the event handling is done by simply reversing the velocity of the bob. For the pendulum to the left a wall is placed directly below the pivot point and the impact occurs when

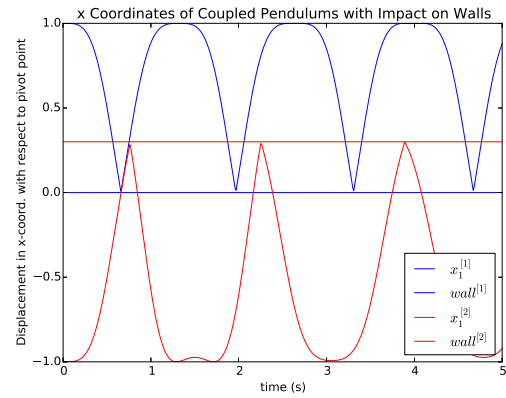


Figure 6. Shows the x-coordinate displacement, with respect to their own pivots, of the pendulums $[1]$, to the left, and $[2]$, to the right, as they hit a wall. The horizontal lines represent walls blocking each pendulums path.

the bobs x-coordinate reaches $\bar{x}_1^{[1]} = 0$ with respect to its pivot. The right-hand-side pendulum wall is placed slightly to the right of its pivot and the bob impacts the wall when its x-coordinate reaches $\bar{x}_1^{[2]} = 0.3$ with respect to its pivot. Initial conditions and parameters are the same as for the example described in Section 3.1.

The aggregated system was integrated with the CVode solver with tolerances $atol = rtol = 10^{-8}$ for 5 seconds. Figure 6 shows the x-coordinate displacement with respect to each pendulums pivot. The two horizontal lines represent each pendulums respective walls.

4 Conclusion

In this paper, a framework has been presented for simulation of coupled systems by aggregation. Care needs to be taken when a coupled system contains feed-through as an equation system needs to be solved in order to compute the derivatives of the system. This puts a condition on the sub-system feed-through terms that also presents itself when computing the Jacobian.

The sub-system events are handled by aggregation. A benefit of this approach is that events from all sub-systems together with external events can be monitored at once and handled through the aggregated system. Example described in Section 3.3 shows that external events can be added to an aggregated coupled system.

The FMI has all functionality needed to carry out the presented scheme. By combining the discussed ideas with Assimulo and allowing direct coupling of FMUs and Python based problems one gets a flexible and powerful environment for solving coupled dynamical problems.

References

- Christian Andersson. *A Software Framework for Implementation and Evaluation of Co-Simulation Algorithms*. Licentiate thesis, Centre for Mathematical Sciences, Lund University, Lund, Sweden, 2013.
- Christian Andersson, Claus Führer, and Johan Åkesson. Assimulo: A unified framework for ode solvers. *Math. Comput. Simulat.*, 2015. doi:10.1016/j.matcom.2015.04.007. In press.
- Torsten Blochwitz, Martin Otter, Johan Åkesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, Hans Olsson, and Antoine Viel. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *In 9th International Modelica Conference 2012*. Modelica Association, 2012.
- Dassault Systèmes. Dymola - Multi-Engineering Modeling and Simulation - Version 2016. <http://www.dymola.com/>, 2016. Accessed: 2015-08-01.
- Edda Eich-Soelner and Claus Führer. *Numerical Methods in Multibody Dynamics*. European Consortium for Mathematics in Industry (ECMI). Teubner, 1998. ISBN 3-519-02601-5.
- Emil Fredriksson, Christian Andersson, and Johan Åkesson. Discontinuities handled with events in Assimulo. In Hubertus Tummescheit and Karl-Erik Årzén, editors, *Proceedings of the 10th International Modelica Conference*, number 96 in Linköping Electronic Conference Proceedings, pages 827–836. Linköping University Electronic Press, Linköpings universitet, 2014. URL <http://dx.doi.org/10.3384/ECP14096827>.
- Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, September 2005. ISSN 0098-3500. doi:10.1145/1089014.1089020.
- Johan Åkesson, Karl-Erik Årzén, Magnus Gäfvert, Tove Bergdahl, and Hubertus Tummescheit. Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problem. *Comput. Chem. Eng.*, 34(11):1737–1749, November 2010. doi:<http://dx.doi.org/10.1016/j.compchemeng.2009.11.011>.

An Aeronautic Case Study for Requirement Formalization and Automated Model Composition in Modelica

Wladimir Schamai¹ Lena Buffoni³ Nicolas Albarello² Pablo Fontes De Miranda² Peter Fritzson³

¹Airbus Group Innovations, Germany wladimir.schamai@airbus.com

²Airbus Group Innovations, France {pablo.fontes-de-miranda, nicolas.albarello}@airbus.com

³IDA, Linköping University, Sweden, {lena.buffoni, peter.fritzson}@liu.se

Abstract

Building complex systems from models that have been developed separately without modifying existing code is a challenging task faced on a regular basis in multiple contexts including design verification. To address this issue an approach has been developed for automating dynamic system model composition by defining the minimum set of information that is necessary to the composition process. In this paper a design and implementation of this approach for standard Modelica is presented in the context of an application case study – the verification of a new design for spoiler activation against requirements.

Keywords: bindings, requirements, model composition, design verification

1 Introduction

Complex cyber-physical systems within safety critical application domains such as avionics need to take a lot of standard and specifications into account (Kepurch, 2010). For complex system, design verification is often challenging due to large number of requirements to be tested. For such systems an automated approach for connecting together system and requirement models is necessary.

Design verification takes place in system development steps starting from early concept evaluation to detailed system component design. The purpose of the presented approach to support design verification activities¹ by automating the task of simulation model composition.

This paper builds upon an approach that enables automated composition of models by expressing the minimum of information necessary to compose the models automatically (Schamai, 2013). In our case study, we show how binding specification can be defined using standard Modelica language (Modelica Association, 2012; Fritzson 2014), and show how the algorithm for automated binding generation can be implemented in OpenModelica. In contrast to an

approach that is based on defining interfaces that models have to implement, this approach enables the integration and/or composition of models without the need for modifying those models. This means that requirement models and system models can be developed separately and existing models can be used without any modifications.

Explicitly exposing and grouping the information that is needed to interconnect the models will reduce analysis work. For example, when several requirements need the same information the same binding specification can be reused.

Additionally, automated generation of binding expressions reduces the risk of introducing errors and reduces modeling effort, in particular in models with highly interrelated components and/or complex binding expressions.

In the case study presented here we wish to verify a particular system design for spoiler activation, represented by a Modelica model, against requirements that are formalized in Modelica using the Modelica Requirements Library (Otter et al, 2014).

This paper is organized as follows: Section 2 presents the case study used in the paper. Section 3 describes the proposed syntax for defining bindings and illustrates it on the case study. Section 4 discusses the implementation of the algorithm for binding generation, and finally Section 5 summarizes the results presented in the paper.

2 Case Study Description

The selected case study is the design verification of the secondary flight system of an aircraft.

The secondary flight control system allows modifying the wing geometry, and consequently the aerodynamic behaviour of the aircraft, during the different flight phases and notably at take-off and landing. It is composed of spoilers, flaps and slats.

¹ Note, since this contribution focuses on implementing of

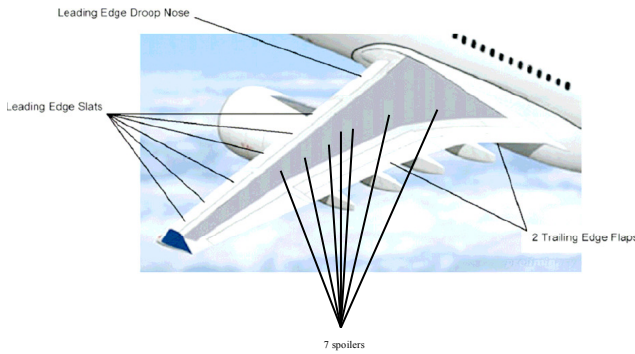


Figure 1. Flight Control System

On the Airbus A350 XWB, some new functions have been attributed to the SFCS system (Strüber, 2014):

- DFS (Differential Flap Setting): possibility of a differential inboard/outboard deflection for loads and drag control
- VC (Variable Camber): Uniform flaps deployment in cruise for drag control
- ADHF (Adaptive Dropped Hinge Flaps): the gaps between the flaps and the spoilers are optimized to reduce turbulences at high and medium speed.

These new functions induced a new architecture of the system and new control logics which both need to be tested.



Figure 2. ADHF configurations

In this new architecture, the actuation of the flaps is done by an actuation chain made of:

- Hydraulic motors
- Electrical motors
- Gears

The actuation of the spoilers is done via actuators being servo controlled actuators (SCA) or Electric Backup Hydraulic Actuators (EBHA). In order to simulate the system behaviour a Modelica model has been developed. The inputs of the system are flap commands, aerodynamic loads on surfaces (flaps and spoilers), and failures of some components. The model essentially uses blocks from the Modelica Standard Library except blocks modeling hydraulic components which were developed specially for this application. For confidentiality reasons, the content of the model cannot be disclosed.

2.1 Requirements Formalization

A system is developed based on requirements which are captured up-front typically using natural language (Hull, 2005). To test requirements they need to be formalized, i.e., they need to be translated into a machine readable form. In our case study we use the new Modelica Requirements Library (Otter et al, 2014) developed in the MODRIO project and the extension for calling blocks as functions implemented in OpenModelica (Buffoni and Fritzson, 2014).

In the following we show some examples of natural language requirements and their corresponding versions in Modelica. Each requirement is modeled such that it explicitly specifies the inputs it requires for evaluation. These inputs will need to be provided by the system or test scenario models. Further, each requirement has an explicit `status` attribute which is the requirement verdict that can take the values *undecided*, *violated* or *satisfied*.

Req.001 “The torque of any ADGB electrical motor shall not be superior to 20 N.m for more than 1 sec.”

This is translated into the following Modelica model.

```

model R1
...
input Torque ADGBtorque = 0;
constant Torque maxTorque = 20;
constant Duration maxDurationForTorqueOvershoot = 1;

Property status(start = Property.Undecided, fixed = true);

Modelica_Requirements.ChecksInFixedWindow.MaxDuration max
xDuration(durationMax=maxDurationForTorqueOvershoot,check=c
ondition.y);
Modelica_Requirements.Sources.BooleanExpression condition(y=
ADGBtorque >= maxTorque);
equation
status = maxDuration.y;
connect(condition.y, maxDuration.condition);
end R1;
    
```

The R1 model has one input `ADGBtorque`. It is the actual torque of any electrical motor. The value will need to be provided the R1 instance by the system model when testing this requirement using simulations.

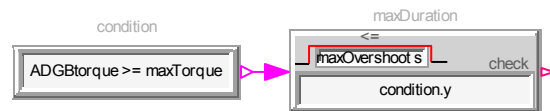


Figure 3. R1 Modelica model

Figure 3 shows the graphical view of the R1 model. It has two components: `Condition` and `maxDuration` from the Modelica Requirements Library. The `condition` component outputs *true* if the actual torque of the motor is greater than the defined threshold and *false* otherwise.

This output is used as input for the `maxDuration` component that outputs the status of the requirement violation. At the very beginning, as long as the condition is false it outputs `undecided`. This is because at this point the requirement was not yet evaluated. As soon as the condition returns `true` the `maxDuration` component will return `violated` if the condition was `true` for longer than 1 sec. or `satisfied` otherwise.

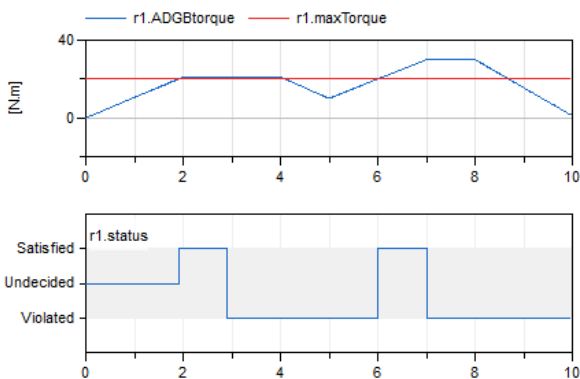


Figure 4. R1 test results

Req.002 “The time of any action of flaps actuation (extension/retraction) shall be less than 50 sec.”.

model R2

```

...
input Boolean isFlapsActuationAction = false;
constant Duration maxDuration = 50;

Property status(start = Property.Undecided, fixed = true);

Modelica_Requirements.ChecksInFixedWindow.MaxDuration ma
xDuration1(durationMax=maxDuration,check=condition.y);
Modelica_Requirements.Sources.BooleanExpression condition(y=i
sFlapsActuationAction);
equation
connect(condition.y, maxDuration1.condition);
status = maxDuration1.y;
end R2;

```

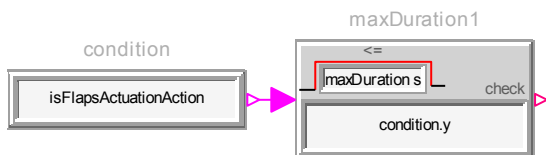


Figure 5. Req 002 Modelica model

The input to this model is `isFlapsActuationAction`. It is a Boolean type value to be provided by the system model when testing this requirement using simulations. Note that at this point it is not clear how to determine whether the flaps actuation action takes place.

In fact there may be several ways of accessing this information of one system design model, and there

may be several system design alternative models. It is the task of the person who develops the design models to specify how this data can be accessed. Section 3 discusses how this can be done.

Figure 5 shows the graphical view of the R2 model. It includes two components: `condition` and `maxDuration` which are instances of models from the Modelica Requirements Library. The `condition` component outputs `true` as long as the action flaps actuation action takes place (i.e., extension or retraction) and `false` otherwise. This output is used as input for the `maxDuration` component that outputs the status of the requirement violation.

At the very beginning, as long as the condition is false it outputs `undecided`. This is because at this point the requirement was not yet evaluated at all. As soon as the condition the flaps actuation action starts, the `maxDuration` component will measure the time. It returns `satisfied` if the action took less than 50 sec. and `violated` otherwise.

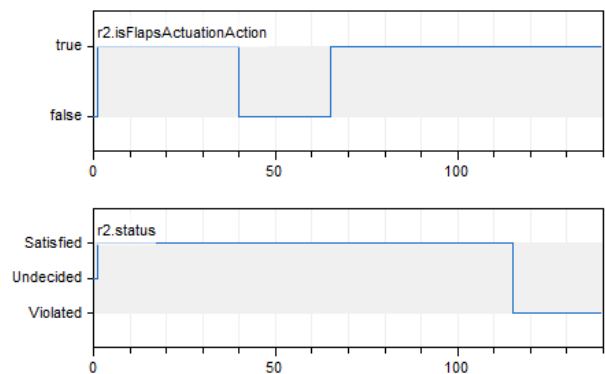


Figure 6. Req 002 test results

Req.003 “The flap angle shall be comprised in the range [-5°;35°]”.

model R3

```

...
input Angle flapAngle;

Property status(start = Property.Undecided, fixed = true);

Modelica_Requirements.LogicalBlocks.WithinBand band1(u_max
=35, u_min=-5, u=flapAngle);
equation
if (not band1.y) then
status = Property.Violated;
else
status = Property.Satisfied;
end if;
end R3;

```

The input for the model R3 is the `flapAngle`. Since there will be several flaps this requirement will need to be checked (i.e., instantiated) for each flap. The model `WithinBand` from the Modelica Requirements library is used for computing the verdict for this requirement.

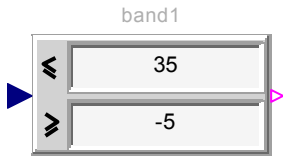


Figure 7. Req 003 Modelica model

The status can be evaluated at any time right from the beginning, i.e., there will be no time instant at which the status is undecided.

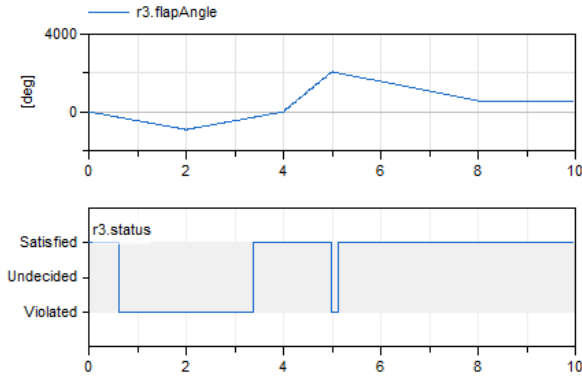


Figure 8. Req 003 test results

Req.004 “When the flap is moving, the distance (gap) between a flap and its spoiler shall be less than 10 cm”.

Req.005 “When the flap is not moving, the distance (gap) between a flap and its spoiler shall be less than 3 cm”.

The formalization of the requirements **Req.004** and **005** is similar to **Req.006** (see below).

Req.006 “The effort between a flap and its spoiler shall be less than 1000N”.

model R6

```

...
input Force forceBetweenFlatAndItsSpoiler=0;
constant Force maxAllowedForce=1000;

Property status(start = Property.Undecided, fixed = true);

Modelica_Requirements.LogicalBlocks.LessThreshold l(threshold
=maxAllowedForce, u = forceBetweenFlatAndItsSpoiler);

equation
if not l.y then
status = Property.Violated;
else
status = Property.Satisfied;
end if;
end R6;

```

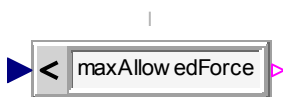


Figure 9. Req 006 Modelica model

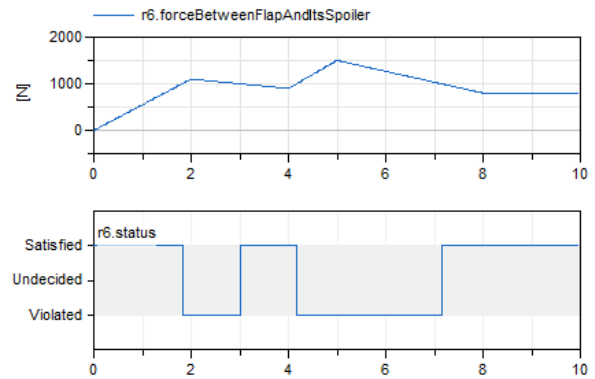


Figure 10. Req 006 test results

Req.015 “The high lift system shall be able to hold the high lift surfaces in their current position:

- Under all load conditions;
- Under all relevant environmental conditions;
- After total loss of electric and hydraulic power (permanent or transient).

model Requirement_15

```

...
input Boolean hydraulicFailure;
input Boolean electricalFailure;
input Angle outboardValue;
input Angle inboardValue;

parameter Real minDerivative = 0.01 "Values in degrees/s";
Property status(start = Property.Undecided, fixed = true);

Modelica_Requirements.ChecksInFixedWindow.During during1(ch
eck=not (flapsMoving.y));
Modelica_Requirements.Sources.BooleanExpression totalFailure(y
= hydraulicFailure and electricalFailure);
Modelica_Requirements.Sources.BooleanExpression flapsMoving(
y=abs(der( SI.Conversions.to_deg(outboardValue))) > minDerivat
ive or abs(der( SI.Conversions.to_deg(inboardValue))) > minDerivat
ive);

equation
status = during1.y;

end Requirement_15;

```

Req.016 “Transients in normal system operations and in case of failure shall not cause excessive loads to components.”

This requirement is quite challenging to formalize since the conditions of excessive loads for each component must be defined.

The following formalization of the requirement uses arrays to gather variables coming from the different instances of the different components (ADGBs, flaps, PCU brakes). The binding mechanism will feed these inputs with the corresponding variables depending on the number of instances present in the model. The *PropertyAnd* block synthesizes the values of the different statuses with a 3-valued “and” logic.


```

model Requirement_16
...
input Angle flapSpeed[:]; //left and right, inboard and outboard
input Real ADGB_MotorTorque[:]; //left and right
input Real ADGB_BrakeTorque[:]; //left and right
input Real PCU_BrakeTorque[:]; //green and yellow brakes

parameter Torque maxMotorTorque = 20;
parameter Real maxDerivative = 2;
parameter Torque maxADGB_BrakeTorque = 1e6;
parameter Torque maxPCU_BrakeTorque = 1e6;

Property status(start = Property.Undecided, fixed = true);

Property ADGB_MotorStatus[size(ADGB_MotorTorque,1)];
Property ADGB_BrakeStatus[size(ADGB_BrakeTorque,1)];
Property PCU_BrakeStatus[size(PCU_BrakeTorque, 1)];
Property flapOverspeedStatus[size(flapSpeed, 1)];

Modelica_Requirements.LogicalBlocks.PropertyAnd andStatus(nu
= size(ADGB_MotorTorque,1)+size(ADGB_BrakeStatus, 1)+size(P
CU_BrakeStatus, 1)+size(flapOverspeedStatus, 1));

equation
andStatus.u = cat(1,ADGB_MotorStatus,ADGB_BrakeStatus,PCU
_BrakeStatus,flapOverspeedStatus);
andStatus.y = status;

for i in 1:size(ADGB_MotorTorque,1) loop
if abs(ADGB_MotorTorque[i])>maxMotorTorque then
ADGB_MotorStatus[i]=Property.Violated;
else
ADGB_MotorStatus[i]=Property.Satisfied;
end if;
end for;

... (same for ADGB_BrakeTorque, PCU_BrakeTorque and
flapSpeed)

end Requirement_16;

```

Req.032 “A single electrical failure shall not prevent an inboard flaps only movement.”

```

model Requirement_32
...
input Boolean electricalFailure;
input Boolean hydraulicFailure;
input Angle outboardValue;
input Angle inboardValue;
input Integer mode; //mode as computed by SFCC

Property status(start = Property.Undecided, fixed = true);

parameter Real minDerivative = 0.01 "Value in rad/s";

Boolean inboardMovement = abs(der(inboardValue))>= minDeriv
ative;
Boolean outboardMovement = abs(der(outboardValue))>= minDer
ivative;

equation
if (mode == 2 and electricalFailure) then //mode 2 = Inboard Differ
ential Flap Setting
if (inboardMovement and not
(outboardMovement)) then
status = Property.Satisfied;
else
status = Property.Violated;
end if;
end if;
end Requirement_32;

```

```

end if;
else
status = Property.Undecided;
end if;
end Requirement_32;

```

Figure 11. Req 032 Modelica Model

In this formalization, a mode computed by the main control computer is used to check if a “inboard flap only movement” is commanded (mode 2). Other implementation could be possible but this one was chosen for its simplicity.

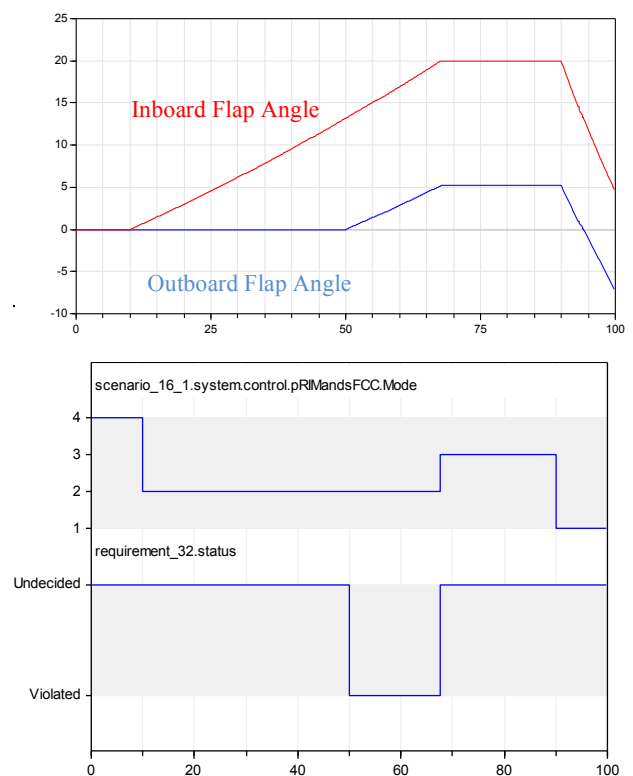


Figure 11. Req 032 test results

The last figure shows the results of a real scenario of system utilization. The model was excited with a pulse entry with 20 degrees of amplitude to move the inboard flaps, with no commands given to the outboard flaps. Also, during the simulation there are cases of an electrical failure distributed in a pulse form.

The requirement is violated during the simulation of the model since the outboard flaps continue to move during the electrical failures. This is due to an error in the model or in the system design and shall be investigated.

2.2 Verification Scenario Formalization

Scenarios are defined to stimulate the system in different conditions. These scenarios are defined as Modelica models providing inputs to the system model (flap commands, loads, failures...).

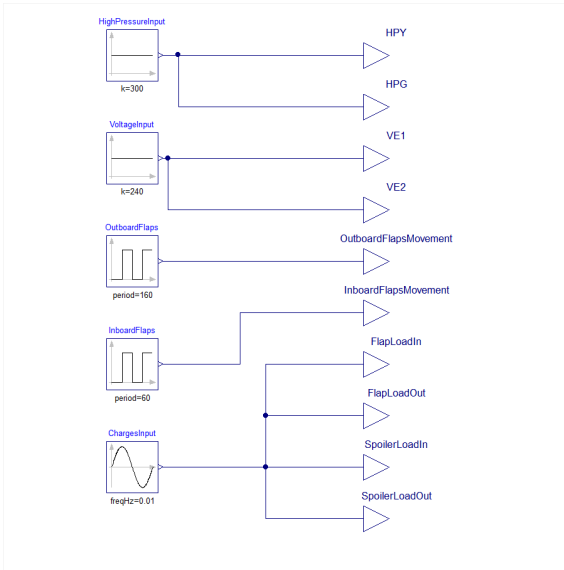


Figure 12. Modeling of a scenario

The scenarios must be defined so that requirements are verified (i.e. some scenarios permit the verification of the requirement). For this, a verification table can be created. This table defines which scenario should permit the verification of which requirements. After simulation of all scenarios, the value of the requirement will permit to check if the table is correct (i.e. to check if the scenarios have triggered the verification of the requirements).

	Sc01	Sc02	...	Sc15	Sc16
R001	X	X			
R002				X	X
...					
R032				X	X

Table 1. Verification table

3 Binding definition

This section describes the syntax for specifying the mediators and generating the bindings. In contrast to our previous proposals for representing bindings which relied either on either the use of an XML representation or on extensions to the Modelica language, the proposal presented in this paper is fully compliant with standard Modelica and relies on records to represent the binding information.

Clients, in this case requirements, require certain data. A record for representing the client, specifies the information necessary from the client side:

```
record Client "Client is a model or component that requires a modifier (i.e. a binding)"
  extends Modelica.Icons.Record;

  String id "A qualified name for the client";
```

```
String template = "" "A transformation that can be applied to the generated binding expression for this client. If left empty, no transformation will be applied.";
```

```
Boolean isMandatory = false "Defines if the client must be bound or if a binding is optional.";
end Client;
```

A number of fields that are optional have predefined values, so that they do not need to be specified if not relevant for a specific binding.

Providers make data available to clients. The information specified by a provider is defined in the record below:

```
record Provider "Provider specifies how to access data required for clients that are linked to the mediator this provider is used for."
  extends Modelica.Icons.Record;
```

```
String id "A qualified name for the provider.";
String template = "" "Code snippet with placeholders used for generating part of binding expression. If left empty, no transformation will be applied.";
```

```
end Provider;
```

Clients and providers do not know each other a priori. In order to relate a set of clients and a set of providers, we use the mediators, defined by the record below:

```
record Mediator "Mediator captures data required for inferring binding expression for referenced clients using referenced providers."
  extends Modelica.Icons.Record;
```

```
String name = "" "Reflects what is needed by referenced clients. Optional.";
String mType = "" "Reflects the type required by referenced clients. Optional.";
```

```
String template = "" "A transformation that can include calls to functions that can handle unsorted arrays (e.g., add(), max(), toArray(), etc.). If left empty, no transformation will be applied.";
```

```
Client clients[:] "List of clients.";
Provider providers[:] "List of providers.";
```

```
end Mediator;
```

A more detailed description of the mediator concept can be found in (Schamai, 2013).

3.1 Binding Specification

Section 2.1 shows examples of formalized requirements. The corresponding requirement models from require the following data:

- Current distance between flap and its spoiler (for R4.distanceBetweenFlapAndItsSpoiler and R5.distanceFlapSpoiler)
- Current flap angle (for R3.flapAngle)
- Current force between flap and its spoiler (for R6.forceBetweenFlapAndItsSpoiler)

- *Current torque of electrical motor* (R1.ADGBtorque)
- *Flap is moving* (for R4.isFlapMoving and R5.isFlapMoving)

The system model determines which of the requirements will be tested and how they will be combined with scenario models. Furthermore, there might be requirements which are repeatedly imposed on system parts of the same kind that exist inside the system model (e.g., there are several flaps in our model).

The purpose of the binding specification is to capture the minimum information in order to enable creating any combination of the system model and a set of requirements such that they will be bound correctly in an automated fashion, as well as to enable determining how many times a particular requirement needs to be instantiated.

In order to do so, the user will now define *mediators* (Schamai et al, 2014). In our example the mediators reflect (i.e., contain information about) what data will need to be provided by the system model in order to enable testing of particular requirements (the clients).

Consider the mediator M1. It defines that any instance of the requirement models R4.distanceBetweenFlapAndItsSpoiler and R5.distanceFlapSpoiler (*clients*) have to be bound² to some other components in order to retrieve the value during simulating. The value can be accessed inside the instance of the type Spoilers.Spoiler_SC.elastoGap (*provider*) by using its sub-component elastoGap.s_rel (captured by the template attribute) whereby getPath() will be replaced by the instance path of the *provider* model. Other mediators are defined in a similar way.

```
record M1
import BindingDefinition.*;
import Req.*;
import SpoilerActuation_v7.*;

extends Mediator(
  name = "Current distance between flap and its spoiler",
  mType = "Modelica.SIunits.Distance",
  clients = {
Client(id="R4.distanceBetweenFlapAndItsSpoiler",
isMandatory=true),
Client(id="R5.distanceFlapSpoiler",
isMandatory=true)},
  providers = {
Provider(id="Spoilers.Spoiler_SC.elastoGap",
template="getPath().elastoGap.s_rel");}
end M1;

record M2
...
extends Mediator(
  name="Current flap angle",
```

```
mType="Modelica.SIunits.Angle",
clients={Client(id="R3.flapAngle", isMandatory=true)},
providers={Provider(id="Flaps.Flap.FlapAngle");});
```

```
end M2;
```

```
record M3
```

```
...
extends Mediator(
  name="Current force between flap and its spoiler",
  mType="Modelica.SIunits.Force",
  clients={Client(id="R6.forceBetweenFlatAndItsSpoiler",
isMandatory=true)},
  providers={Provider(id="Spoilers.Spoiler_SC.elastoGap",
template="getPath().flange_a");});
```

```
end M3;
```

```
record M4
```

```
...
extends Mediator(
  name="Current torque of electrical motor",
  mType="Modelica.SIunits.Torque",
  clients={Client(id="R1.ADGBtorque", isMandatory=true)},
  providers={
Provider(id="Flaps.ActuationChainComponents.MotorModel.flange_b",
template="getPath().tau");});
```

```
end M4;
```

```
record M5
```

```
...
extends Mediator(
  name=" Flap is moving",
  mType="Boolean",
  clients={
Client(id="R4.isFlapMoving", isMandatory=true),
Client(id="R5.isFlapMoving",isMandatory=true)},
  providers={
Provider(id="Control.SFCC.Mode", template="getPath() <> 4")
});
```

```
end M5;
```

```
record M6
```

```
...
extends Mediator(
  name="Flaps actuation action is taking place",
  mType="Boolean",
  clients={Client(id="R2.isFlapsActuationAction",
isMandatory=true)},
  providers={Provider(id="Control.SFCC.Mode",
template="getPath() <> 4");});
```

```
end M6;
```

4 Binding generation

Once the bindings are specified a verification model can be created containing the system model and the requirements to be verified.

```
model VeM01
import Req.*;
import SpoilerActuation_v7.*;
System sm_system;
R1 r1; R2 r2; R3 r3; R4 r4; R5 r5; R6 r6;
end VeM01;
```

² This is indicated by the attribute `isMandatory=true`

The system model imports the packages where the mediators that can be used in the binding computation are defined.

The OpenModelica API has been extended with a call: `inferBindings(systemModel, program);`

The call accepts as arguments the name of `systemModel` as well as the environment (here `program`) with all the loaded classes where it will look for the mediator definitions and update the `systemModel` with the binding expressions in the form of modifiers.

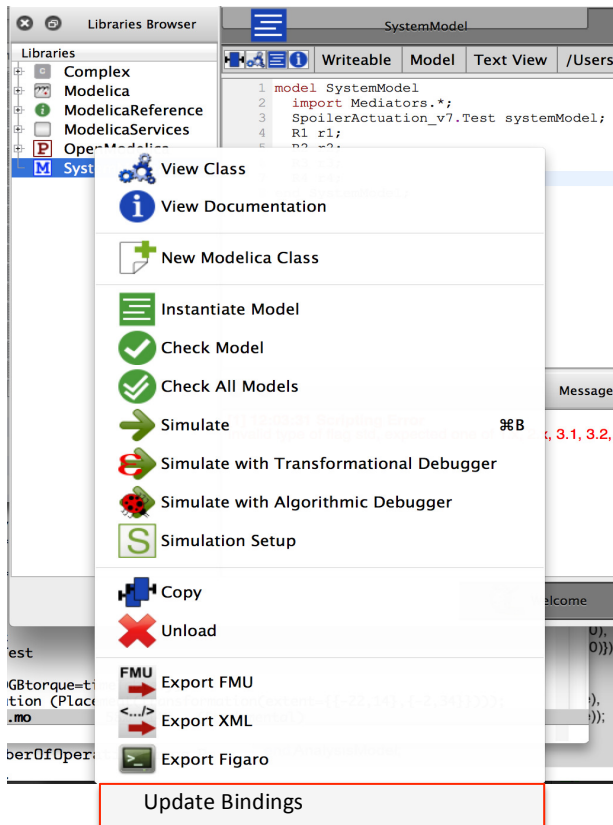


Figure 13 Binding generation in OpenModelica

The algorithm for binding generation is implemented in OpenModelica as it is defined in (Schamai et al, 2014). First an instance tree is built for the model to be bound (see Figure 14). This instance tree is represented in an internal data structure and all the clients and providers are identified by checking whether they match the client or provider paths defined in any mediators. For instance mediator M4 specifies only one client : `Client(id="R1.ADGBtorque", isMandatory=true)` and therefore ADGBtorque will be marked as a client in the instantiation tree. All the mediator data is also stored in an internal structure with references to all the instances of clients and providers found for each mediator.

Once all the internal structures are created, for each client the bindings are computed by localizing all the providers. If more than one provider is present then a template must be defined in the mediator to describe how the inputs from different clients must be combined. In mediator M4 we only have one provider defined:

`Provider(id="Flaps.ActuationChainComponents.MotorModel.flange_b", template="getPath().tau")`

As in the model we have two instances of `MotorModel`, right and left, two provider instances will be found by the algorithm.

The `getPath()` call is replaced with the path of the component instance in the environment, in this example

`sm_system.flaps.actuationChain.ADGB_Left.motorModel.flange_b` and `sm_system.flaps.actuationChain.ADGB_Right.motorModel.flange_b` and the template is used to generate a binding expression, in this case to point to `tau` inside each of the providers.

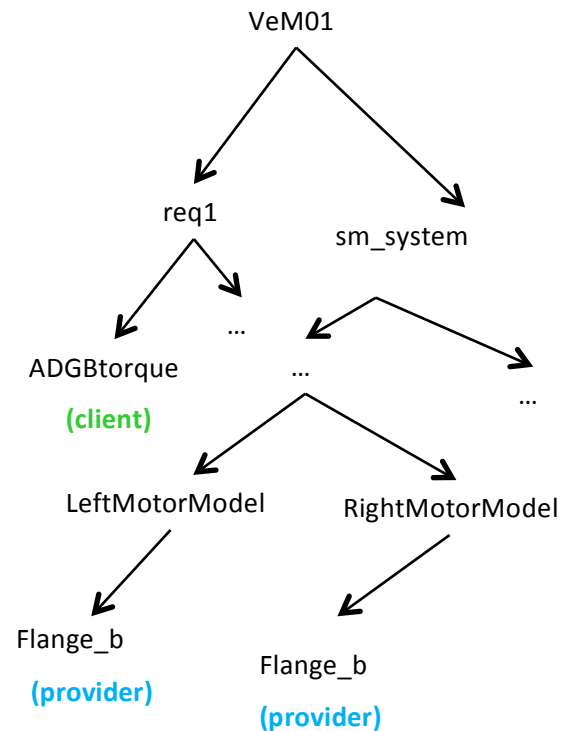


Figure 14 Instantiation tree

The algorithm figures out the required number of requirement instantiations and generates the binding expressions. For instance, in this example we need two instances of R1, one for each motor and as the model used in this use case has four flaps, four instances of R4 will be needed. Binding expressions are implemented as modifiers that will be applied to the components and sub-components of `systemModel`.

If the algorithm cannot find a binding for a mandatory client, or several binding are possible for the same client, the result will be an error message.

Similarly, when several instances of requirements that require more than one input need to be generated, user involvement may be needed to indicate how to correctly pair up the providers. In future versions of the implementation, support for storing and reusing user decisions will be implemented through the use of annotations.

For example, regarding the system model specified for this case study, the algorithm will generate the following binding expressions:

```

model VeM01
import Req.*;
import SpoilerActuation_v7.*;

System sm_system;
R1 req_001_0_r1(ADGBtorque = sm_system.flaps.actuationChain.
  ADGB_Left.motorModel.flange_b.tau);
R1 req_001_1_r1(ADGBtorque = sm_system.flaps.actuationChain.
  ADGB_Right.motorModel.flange_b.tau);
R2 req_002_r2(isFlapsActuationAction = sm_system.control
  .pRIMandsFCC.Mode <> 4);
R3 req_003_0_r3(flapAngle = sm_system.flaps.FlapLI.FlapAngle);
R3 req_003_1_r3(flapAngle
  = sm_system.flaps.FlapLO.FlapAngle);
R3 req_003_2_r3(flapAngle = sm_system.flaps.FlapRI.FlapAngle);
R3 req_003_3_r3(flapAngle
  = sm_system.flaps.FlapRO.FlapAngle);
R4 req_004_0_r4(distanceBetweenFlapAndItsSpoiler =
  sm_system.LISpoiler.elastoGap.elastoGap.s_rel,
  isFlapMoving =
  sm_system.control.pRIMandsFCC.Mode <> 4);
R4 req_004_1_r4(distanceBetweenFlapAndItsSpoiler =
  sm_system.LOSpoiler.elastoGap.elastoGap.s_rel,
  isFlapMoving =
  sm_system.control.pRIMandsFCC.Mode <> 4);
R4 req_004_2_r4(distanceBetweenFlapAndItsSpoiler =
  sm_system.RISpoiler.elastoGap.elastoGap.s_rel,
  isFlapMoving =
  sm_system.control.pRIMandsFCC.Mode <> 4);
R4 req_004_3_r4(distanceBetweenFlapAndItsSpoiler =
  sm_system.ROSpoiler.elastoGap.elastoGap.s_rel,
  isFlapMoving =
  sm_system.control.pRIMandsFCC.Mode <> 4);
R5 req_005_0_r5(distanceFlapSpoiler =
  sm_system.LISpoiler.elastoGap.elastoGap.s_rel,
  isFlapMoving =
  sm_system.control.pRIMandsFCC.Mode <> 4);
R5 req_005_1_r5(distanceFlapSpoiler =
  sm_system.LOSpoiler.elastoGap.elastoGap.s_rel,
  isFlapMoving =
  sm_system.control.pRIMandsFCC.Mode <> 4);
R5 req_005_2_r5(distanceFlapSpoiler =
  sm_system.RISpoiler.elastoGap.elastoGap.s_rel,
  isFlapMoving =
  sm_system.control.pRIMandsFCC.Mode <> 4);
R5 req_005_3_r5(distanceFlapSpoiler =
  sm_system.ROSpoiler.elastoGap.elastoGap.s_rel,
  isFlapMoving =
  sm_system.control.pRIMandsFCC.Mode <> 4);
end VeM01;

```

Once the bindings are defined, if we want to modify the system design, for instance adding backup components to the system or modifying the number of flaps, then the bindings can be regenerated with no additional effort.

Moreover, bindings can be used in batch testing to automatically generate verification models with different scenarios and different requirement subsets. This is something that would be difficult to do using explicit interfaces.

5 Conclusion

In this paper we have presented:

- A new application of design verification on an industrial case study in the field of aeronautics.
- The use of the new requirement modeling library for formalizing the requirements of the case study. We have shown that the binding approach is fully compatible with the new Modelica Requirements library.
- A modified version of the syntax for representing binding specification that is fully compliant with standard Modelica syntax, meaning that binding specifications can be edited and visualized in any Modelica tool. In order to support the binding generation, a tool has to simply implement the binding algorithm in (*Schamai, 2014*).
- An implementation of the binding algorithm in OpenModelica

The binding approach does not assume prior knowledge of each other by the respective models and therefore increases decoupling and allows reuse of existing models and libraries. As mediators can be defined in several steps this means that different people can provide the information necessary to connect the models at different stages in the design process.

Moreover, the binding algorithm is general and can be used for binding models in other contexts than requirement verification. Furthermore, it enables a formal traceability between client and provider models. For example, determining which requirements are implemented in the system design model at hand can be achieved by looking at the bindings for mandatory requirement clients.

The case study is work in progress, but it has already allowed to detect a number of relevant issues in the model.

Clearly, the effectiveness of the presented approach is jeopardized when bindings are specified such that they result into too many ambiguous matches to be resolved by user manually. Such situation should be detected. Possible resolution could include: Providing hints for modifying the binding specifications; Enabling user to add more information to the binding specification for handling special cases; Or supporting the user by providing the list of all possible combinations to choose from. More complete results, for example, the evaluation of this approach on real projects with large number of requirements is still subject to future work.

Acknowledgements

This work is partially supported by the EU INTO-CPS project and the ITEA 2 MODRIO project via the Swedish Government (Vinnova) and the German and French Government.

References

- Lena Buffoni and Peter Fritzson. Expressing Requirements in Modelica. In *Proceedings of the 55th International Conference on Simulation and Modeling (SIMS 2014)*, Aalborg, Denmark, October 21-22, 2014.
- Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. 1250 pages. ISBN 9781-118-859124, Wiley IEEE Press, 2014.
- Hull, E., Jackson, K., and Dick, J. *Requirements Engineering*. Springer, 2005.
- Kapurch, S. NASA Systems Engineering Handbook. DIANE Publishing Company, 2010. URL <http://books.google.se/books?id=2CDrawe5AvEC>.
- Martin Otter, Lena Buffoni, Peter Fritzson, Martin Sjölund, Wladimir Schamai, Alfredo Garro, Andrea Tundis, Hilding Elmqvist. D2.1.1 – Modelica Extensions for Properties Modelling, Part IV: Modelica for Properties Modeling. Internal Report, ITEA2 MODRIO project, Sept. 2014.
- Modelica Association. Modelica, A Unified Object-Oriented Language for Systems Modeling, Language Specification, Version 3.3, May 9, 2012. <https://www.modelica.org/documents/ModelicaSpec33.pdf>
- Wladimir Schamai. *Model-Based Verification of Dynamic System Behavior against Requirements*. Ph.D. thesis, Method, Language, and Tool Linköping: Linköping University Electronic Press, Dissertations, 1547, 2013..
- Wladimir Schamai, Lena Buffoni, and Peter Fritzson, An Approach to Automated Model Composition Illustrated in the Context of Design Verification. *Journal of Modeling, Identification and Control*, Volume 35- 2, pages 79—91, 2014.
- H. Strüber The Aerodynamic Design of the A350 XWB-900 High Lift System. 29th Congress of the International Council of the Aeronautical Sciences. St Petersburg, 2014.

FastHVAC - A library for fast composition and simulation of building energy systems

Sebastian Stinner Markus Schumacher Konstantin Finkbeiner Rita Streblow
Dirk Müller

RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Aachen, Germany

{sstinner, mschumacher}@eonerc.rwth-aachen.de

Abstract

This paper describes the implementation of a Modelica library that is designed to enable fast composition and simulation of building heating, ventilation and air conditioning (HVAC) systems. The library is based on an approach which is focusing on the thermal behavior of the components. Compared to existing libraries, it has no information about the pressure of the system. This approach limits the applicability of the library, but decreases the computational effort as well as the time to set up a model. Nonetheless, the simulation results are comparable to simple HVAC system models based on `Modelica.Fluid`.

Keywords: HVAC, building performance simulation, building energy systems, city district simulation

1 Introduction

The dynamic simulation of building energy systems, including HVAC technologies, is gaining importance in recent years. Various open source libraries for the simulation of HVAC systems are available today (AixLib, Buildings, BuildingSystems, IDEAS) (Fuchs et al., 2015; Wetter et al., 2014; Nytsch-Geusen et al., 2013; Baetens et al., 2015) and will be unified in the context of the IEA EBC Annex 60 activities (Annex60, 2015; Wetter et al., 2015). The applications of the simulation models range from the rapid prototyping of new components and systems, over development of advanced control systems to reuse of models during operation for fault detection and diagnostics (Wetter, 2009).

In recent years, applications for BES system models have been extended to analyze entire city districts (Müller et al., 2015). This is especially important, when considering the interconnection of buildings, which can be in terms of district heating networks or the electrical grid. In this field, measures like Demand Side Management (DSM) (Müller et al., 2015) are expected to become more important in the future. The simulative analysis of

these (large) energy systems with implemented thermal storages, as well as heat generators, e.g. heat pumps and combined heat and power plants, requires the analysis of large equation systems and might lead to unreasonable computational effort.

Furthermore, the simulation of (especially closed) hydraulic circuits, whose models are based on the package `Modelica.Fluid` of the Modelica Standard Library (MSL) shows considerable difficulties. In particular, the steady-state initialization of these systems can be a critical issue. Singularities in the solution can arise. These singularities do not occur in single component testing, but in the composition of the closed hydraulic cycle. For inexperienced users, this behavior can be confusing (Casella, 2012). Even if the aforementioned phenomenon is encountered by experienced users, it might be a challenging task to set the initialization values properly. Especially in cases where the thermal investigation of energy systems is focused, it requires additional expenditure without benefit. Furthermore, the computational effort can be a critical factor, depending on the size of the observed system. The number of initialization variables can grow very fast (Casella et al., 2011). In case of a city district analysis, this becomes a very critical issue, as the building energy systems should probably be parameterized automatically with little manual input. As most of the critical problems in HVAC simulations arise due to pressure calculations, the authors developed an additional library for the simulation of building energy systems that takes into account the whole thermal behavior, while reducing the information about the hydraulic circuit to the mass flow rate. The introduced library is particularly suitable for applications such as rapid prototyping of new energy systems and the development of advanced control systems for heat generators.

In the following, we will introduce the approach that forms the basis for all simulation models in the presented library. First, all required equations and base classes are introduced. Afterwards, we give an overview of the library itself and the included packages with a selection of the models. The developed library enables the user

for fast composition and simulation of building energy systems. However, the accuracy of the simulated models has to be verified. This is shown in chapter 4 by a comparison of a building energy system modeled with both the `Modelica.Fluid` components and the components of the new library, called `FastHVAC`.

2 Modelling principles

The models of the `FastHVAC` library are designed based on standard mass and energy balances in individual components. Each component includes a volume, whose temperature, specific heat capacity (thus also enthalpy) and density is considered to be homogenous. To have the ability to interconnect different components, a new connector, that only transports the required information between the components, has been developed. This connector is called `EnthalpyPort`. It carries the following information:

- the mass flow rate in kg/s: designed as a flow variable, from the connection point into the component
- the thermodynamic temperature in K
- the specific enthalpy of the fluid in J/kg as a stream variable
- the (constant) specific heat capacity of the fluid in J/(kg K)

Here, the specific heat capacity is a redundant information. It is needed for development purposes of the `FastHVAC` library. After completion of the development phase, the variable will be removed. With known temperature difference, mass flow rate and the specific heat capacity, it is possible to calculate the heat flow at each time step. Considering the component's thermal behavior, it facilitates the fault detection of models.

To make use of the aforementioned variables, balance equations for each component are formulated in accordance with Figure 1.

$$\sum \dot{m}_i = 0 \quad (1)$$

This equation defines that the fluid mass in the components does not change over time and entering mass flows have to leave through another connection.

Furthermore, the energy balances are formulated:

$$c_1 = c_2 = c \quad (2)$$

$$T_2 = \text{heatPort}.T; \quad (3)$$

$$h_2 = c \cdot T_2 \quad (4)$$

$$\dot{Q} = \dot{m} \cdot (h_2 - h_1) \quad (5)$$

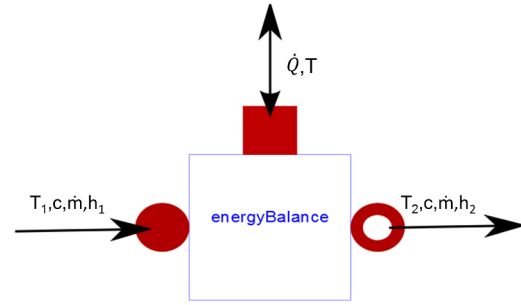


Figure 1. Exemplary scheme of the model `EnergyBalance`

The model `EnergyBalance` is based on the equations (1) to (5) and forms the basis for further components. Here, for entering flows index 1 is used and index 2 for outgoing flows, see Figure 1.

In case of hydraulic networks with multiple branches it is necessary to split off and merge the mass flows. Since there is no temperature change, the splitting of mass flows is described by the standard mass balance:

$$\dot{m}_{in} = \sum_{i=1}^n \dot{m}_{out,i} \quad (6)$$

$$T_{in} = T_{out} \quad c_{in} = c_{out} \quad h_{in} = h_{out} \quad (7)$$

Merging the entering mass flows with different temperatures requires a calculation of the outgoing temperature. The temperature of the outgoing flow is obtained by balancing the mass and enthalpy flows:

$$\dot{m}_{out} = \sum_{i=1}^n \dot{m}_{in,i} \quad (8)$$

$$\dot{H}_{out} = \sum_{i=1}^n \dot{H}_{in,i} \quad (9)$$

$$T_{out} = \frac{\dot{H}_{out}}{c_p \cdot \dot{m}_{out}} \quad (10)$$

If a certain heat capacity is available in the component due to fluid content, the component `HeatTransfer.Components.HeatCapacitor` from the package `Modelica.Thermal` of the MSL, is introduced. Following this approach, we assure that the temperature dynamics within the component are properly represented and the leaving fluid has the actual component fluid temperature.

As many of the components (e.g. heat generators, pipes) have a capacity and need an energy balance as well, a combination of both systems is implemented forming the model `WorkingFluid`. This component is basically equivalent to the `Vessels.ClosedVolume` model of the `Modelica.Fluid` package within the MSL. The `WorkingFluid` model has a `HeatPort_a` connection to the outside, where heat can be injected or removed from the component.

The commonly used fluid in building heating systems in Germany is water. However, within the operating range of standard systems, the fluid properties undergo only marginal changes. Therefore, we assume for the first version of the `FastHVAC` library properties like density, specific heat capacity and thermal conductivity to be constant.

3 Library for building HVAC systems

3.1 Packages of the `FastHVAC` library

The `FastHVAC` library is organized into the packages shown in Figure 2. Most packages contain a package `BaseClasses`, which comprises essential models for the individual components.

```
FastHVAC.BaseClasses.EnergyBalance
                        .WorkingFluid
FastHVAC.Components.HeatGenerators
                        .Storage
                        .Pipes
                        .Pumps
                        .Valves
                        .HeatExchanger
                        .Sources
                        .Sinks
                        .Sensors

FastHVAC.Examples
FastHVAC.Interfaces
FastHVAC.Media
```

Figure 2. Package structure of the `FastHVAC` library, current state. Only the major packages are shown.

3.1.1 Package `BaseClasses`

The package `BaseClasses` contains the models `EnergyBalance` and `WorkingFluid`, which represent the principles described in section 2.

3.1.2 Package `Components`

The classes that form the core of the `FastHVAC` library can be found in the package `Components`. These can be organized into components for generation, storage and distribution of thermal energy. Besides, there are additional packages that enable device control systems by measurements or are necessary to form proper energy balances at the system boundaries. In the following, these individual components will be briefly introduced.

- The generation of thermal energy can be represented by three different technologies, namely a `Boiler`, `CHP` or `HeatPump` component. Additionally, we include the model of a `Solar.Thermal` as a renewable source of heat production, which can be directly coupled to a test reference year (TRY) (DWD, 2011) based weather component.
- The model `Storage` is the implementation of a stratified water storage tank with variable dimensions. The tank loading and unloading can occur either directly via mass flow injection or indirectly via up to two heating coils. The number of connection pairs for direct loading and unloading cycles is variable. The position of the connection pairs is variable as well and can be adjusted to individual purposes. A more detailed description and assessment of this component is shown in section 4.1.3.
- To cover the distribution of produced thermal energy to a room or building, further models are introduced. The model `Pumps` is used to generate a heating fluid mass flow through the system, whereas the model `Pipes` is used to direct the fluid to its destination and provide a physical connection to the ambient (e.g. air or soil). Since no pressure calculations are performed in the `FastHVAC` approach, the fluid mass flow rate is an external input to the `Pump` model and can be ideally set. `Valves` are needed to split off and merge fluid mass flows in multiple circuits. Due to the missing pressure information, the valve functionality is based on external signals. Considering a two-way valve, an equivalent mass flow rate to the valve opening can be calculated, e.g. by using a valve characteristics function. The calculated mass flow rate is used as the external input for the `Pump` model. If one closed loop with two branches is considered, a three-way valve can be used as well as two pumps inside each branch. The three-way valve divides the incoming mass flow rate into two flows. The ratio can be controlled by an external input. In case of more than two branches within one closed loop, a combination of three-way valves and pumps can be used.
- In cases where a conductive heat transfer has to be considered, e.g. in a domestic hot water (DHW) station, the model `DHWHeatExchanger` in the package `HeatExchanger` is applied. To supply the building with heat for space heating, a model for radiators called `Radiator_ML` is incorporated in the package `HeatExchanger`. Several parameters of the heat exchangers, for instance dimensions and discretization, can be adjusted.
- To enable the consideration of non-circular energy flows across the system boundaries, such as the flow of fresh water from public supply into the DHW

system, `Sources` and `Sinks` are available within the library. These models are comparable to the `Modelica.Fluid` components.

- For device control purposes, measurement tools like a temperature sensor and a mass flow sensor are available within the package `Sensors`.

3.1.3 Other packages

The package `Examples` contains example applications for each component that illustrate the typical use, test and validation cases. The package `Interfaces` contains connectors to build interconnections between individual components. This means especially the connector `EnthalpyPort` and the derived connectors. The package `Media` contains data records of different medium models.

4 Example applications

As the models created with the `FastHVAC` approach should not change the physical behavior of a considered system, we compare them to implementations with other models. For comparison purposes, each component is modeled based on `Modelica.Fluid` (HVAC component) as well as on the new approach (`FastHVAC` component). In the following, we will use the `Modelica.Fluid` based approach as reference.

First, different single components will be analyzed separately. In this case, the boiler, the radiator and the heat storage are shown. For each component, a `Modelica.Fluid` based model is compared to a `FastHVAC` model. For the boiler and the radiator, both models have exactly the same functionality and parameters. In the case of the heat storage, an open source model is used for the comparison. We use a model from the `Buildings` library (Wetter et al., 2014) for verification of the `FastHVAC` model. Afterwards, we will present the comparison results of a whole building energy system. It is a heat supply system for a dwelling, which comprises a heat consumer, heat generator, heat exchanger and a pipe network.

4.1 Heat supply components for a dwelling

For verification purposes, we will compare the dynamic thermal behavior of each component in simple test cases. The utilized medium record for `Modelica.Fluid` based components is the simple liquid water with constant properties (incompressible). The record values for `FastHVAC.Media` are identical to `ConstantPropertyLiquidWater` data from `Modelica.Media.Water` within `MSL` (specific heat capacity, density, thermal conductivity). In the following, the different test cases are introduced.

4.1.1 Test case: Boiler

The heat generation is provided by a gas fired boiler. The boiler model is able to vary the flow temperature by modulating operation, e.g. to hold a certain flow temperature in case of varying return mass flow rate or return temperature. The observed time period is set to 1 d and the simulation resolution to 60 s.

In this test case, the flow temperature and mass flow rate are constant while the heating power is increasing linearly from 30 % to 100 % within 10 h. Furthermore, the boiler is switched off for 5 h after every 5 h of operation.

In Figure 3 the simulation results of both approaches are shown. The boiler return temperature is the variable of comparison. For this rather simple test case, the results are identically equal.

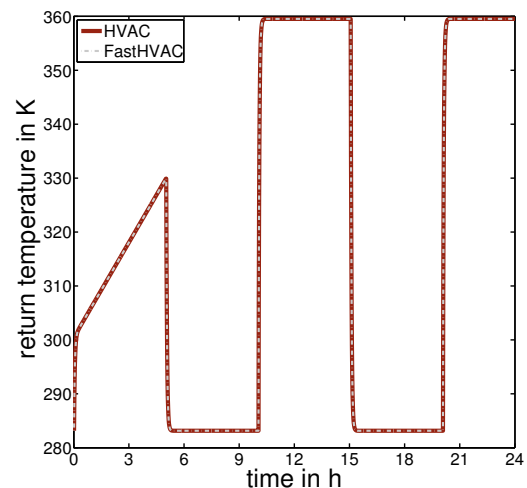


Figure 3. Boiler return temperature

4.1.2 Test case: Radiator

The heat distribution to the dwelling is provided by a radiator and is based on radiative and convective heat transfer. In this case, the thermal output of the radiator is determined by the building heating load. During nighttime, the heating load is higher than during daytime, mainly because of the absence of solar gains. In the considered case, the flow temperature signal is a sine wave with a period of 24 h. This should emulate a dynamic operation of the system. The mass flow rate and ambient temperature are constant. The observed time period is again set to 1 d and the simulation resolution to 60 s.

Target variables are the return temperature and the thermal output of the radiator. The comparative simulation shows identical results, see Figure 4.

4.1.3 Test case: Heat storage

A heat storage is not part of the closed heat supply system considered in section 4.2, but it is an important

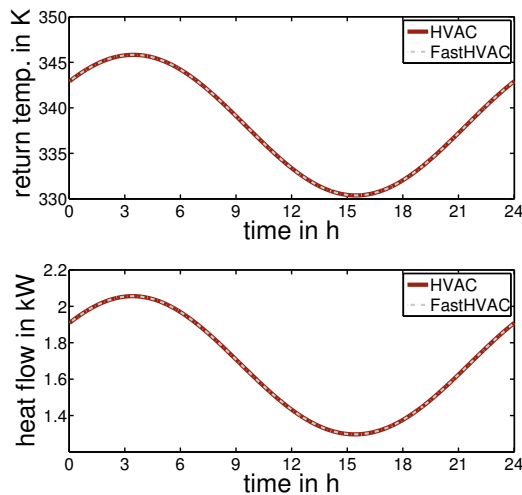


Figure 4. Return temperature and thermal output of radiator

component in general. For instance, in case of DSM, a thermal storage is necessary to decouple heat generation from heating demand. The heat transfer between the individual storage layers is based on conductance, buoyancy and enthalpy flow. Heat losses through the wall are accounted for as well.

The storage model `Fluid.Storage.Stratified` within the `Buildings` library is based on the same heat transfer principles (Wetter, 2015a). Therefore, we compare the storage following the `FastHVAC` approach, to the `Buildings` storage model for verification purposes. It is important to note that the storage wall construction of the considered models differs in its structure. In case of the model `Buildings.Fluid.Storage.Stratified`, the whole storage wall is represented by the model `ThermalConductor` of the package `Modelica.Thermal.HeatTransfer.Components`. In contrast, the storage wall of the `FastHVAC` component consists of a tank wall and a tank insulation. Both, the wall and insulation are represented by a combination of the models `ThermalConductor`, `HeatCapacitor` of the package `Modelica.Thermal.HeatTransfer.Components` and additionally a model for heat convection. All other parameters and model details are identical. Both storage models are parameterized with the same dimensions, five layers and the same characteristic time constant for mixing. The ambient temperature is set constant.

The considered case shows a typical use case of a heat storage tank. The storage is first on standby mode followed by an alternating unloading and loading operation. This operation is modeled as a pulse signal, which is changing the flow temperature. The mass flow rate is modeled as a pulse signal as well. The amplitude changes between zero, in case of standby mode, and the nominal mass flow rate of the loading and unloading cycle.

Variables of interest are the top and bottom layer temperatures and the heat losses. Figure 5 shows the simulated dynamic temperatures of the top and bottom storage layer. The results are comparable, especially during the loading and unloading operation. For clarification, the temperature differences are shown in Figure 5 as well. The differences fade out when coming closer to a steady-state. The coefficient of determination is in both cases nearly 1. Due to different wall constructions, the dynamic heat losses are different, as shown in Figure 6. However, the mean values are in a comparable range, with those of the `Buildings` component slightly higher.

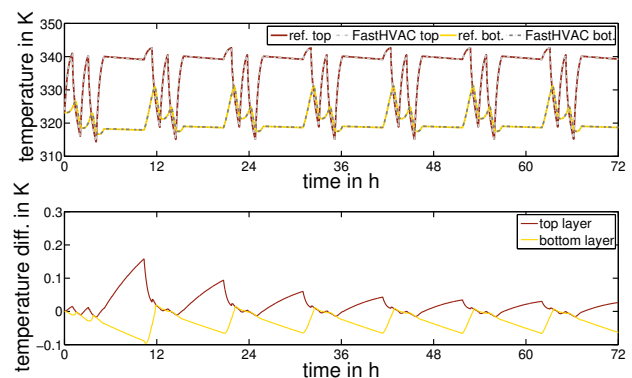


Figure 5. Absolute temperatures and temperature differences of top and bottom storage layers

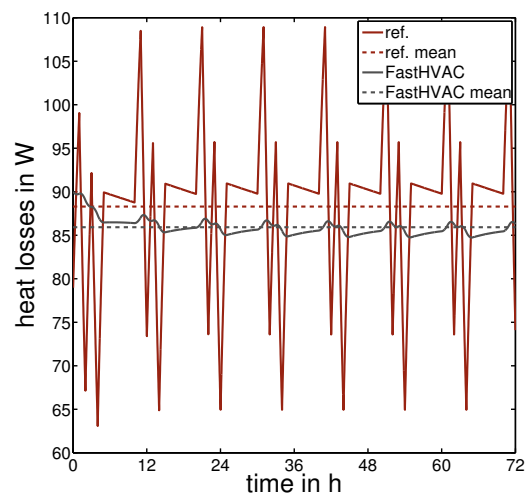


Figure 6. Storage heat losses to ambient

4.2 Heat supply system for a dwelling

The considered model of a dwelling heat supply system following the `FastHVAC` approach is shown in Figure 7. The models `Boiler` and `Radiator_ML` are already introduced. The dwelling as a heat consumer is represented by the model `thermalZone` which is freely available in the open-source model library `AixLib` (Fuchs et al., 2015). This model describes the thermal building behavior under given environmental conditions (Lauster et al.,

2014). In this case, we consider a single-family house with an insulation standard corresponding to the year 2002. The model `weather` represents the environment. Based on a TRY of the Germany's National Meteorological Service, weather data input signals (e.g. ambient air temperature, solar irradiation) are fed into the building model (DWD, 2011). The model `pID` controls the room temperature by varying the fluid mass flow through the boiler. The model `controllerBoiler` controls the radiator flow temperature by adapting the thermal output of the boiler.

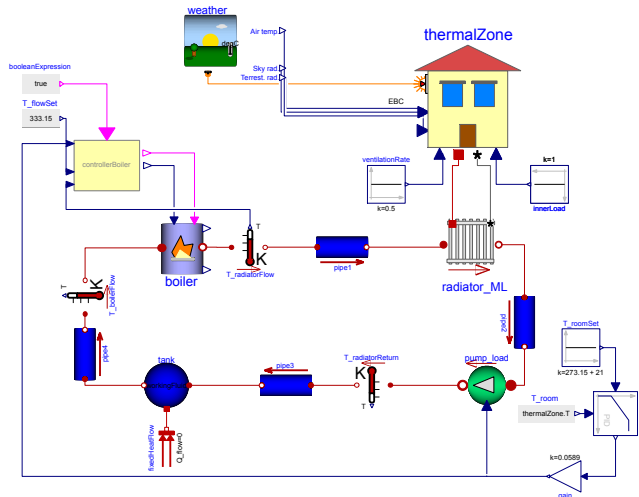


Figure 7. Model of heat supply system

In case of a closed thermo-hydraulic circuit with an incompressible fluid based on `Modelica.Fluid`, a pressure reference is necessary to avoid a structural singularity (Wetter, 2015b). For this reason, an adiabatic tank model is used within the `Modelica.Fluid` based variant. For closed thermo-hydraulic circuits, based on the new approach, these kind of measures are not necessary. However, for the observed case, it is considered for comparison purposes. The additional fluid content which is brought to the system by pipes and the tank has an influence on the dynamic behavior of this system. To take this fluid content into account, the models `tank` and `pipe` which contain the component `workingFluid` are applied. In order to achieve adiabatic conditions for these components, the heat transfer to the environment is set to zero or is disabled.

In this heat supply system, the room temperature is set to 294.15 K. The radiator flow temperature is set to 333.15 K. The simulated time period covers one year and the output resolution is 1 h. Variables of interest are the boiler gas demand, the room temperature and the fluid mass flow.

In case of the fluid mass flow, the simulated results have no significant differences. Figure 8 shows a scatter plot of the dynamic fluid mass flow rate of both approaches. In the range of lower mass flow rates, there is a minor deviation. Nevertheless, the coefficient of deter-

mination is approximately 1.

The deviation of the mass flow rates arises in cases of closed hydraulic circuits. The initial conditions of hydraulics in case of the `Modelica.Fluid` based variant (e.g. `dp_small`, `m_flow_small`, `m_flow_start`, etc.) have an influence on this deviation. The initial hydraulics settings cannot be set randomly due to the initialization difficulties mentioned in the introduction.

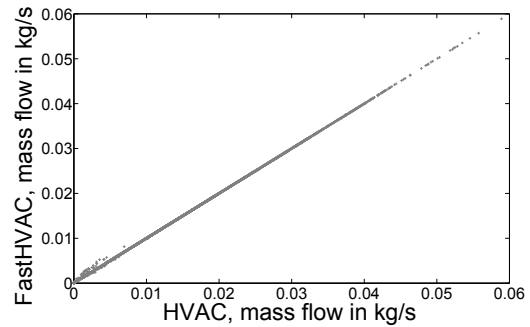


Figure 8. Comparison of fluid mass flow rates

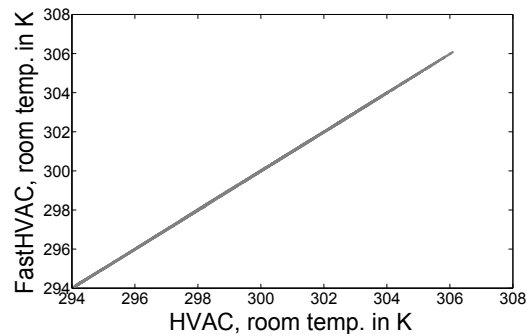


Figure 9. Comparison of room temperatures

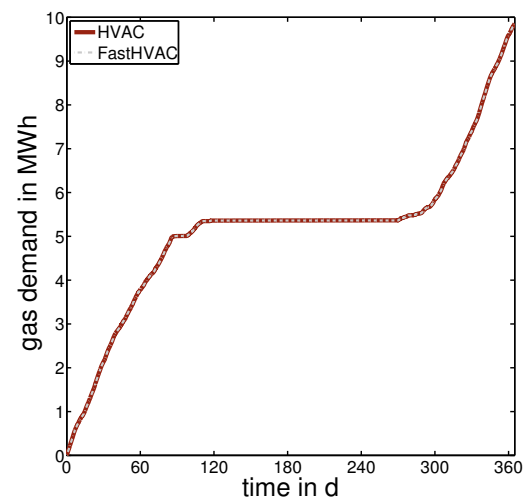


Figure 10. Boiler gas demand

Due to the minor deviation of the mass flow rate, the simulated dynamic room temperatures differ as well. A scatter plot of the simulation results is shown in Figure

9. However, the coefficient of determination is approximately 1. There is no active cooler inside the system. For this reason, the room temperature in the summer period is higher than the room set temperature.

The minor mass flow rate deviation has almost no influence on the boiler gas demand. Figure 10 shows the gas demand for the whole simulation period. In this case, the coefficient of determination is approximately 1 as well, with a negligible error for the annual gas demand.

Having a look at the simulation speed, we see a major improvement with the `FastHVAC` library. In case of the `Modelica.Fluid` based system, we found a computational effort of 379 seconds, compared to 79 seconds following the new approach.

5 Conclusions and work in progress

In this paper, we presented the design of a building HVAC library. This library is particularly suitable for applications such as rapid prototyping of innovative energy systems and the development of advanced control systems for heat generators. It is designed to decrease computational effort, while maintaining the result quality of more complex libraries. As stated in the case study, the simulation speed can be increased noticeably following our new approach. Nevertheless, the results are almost identical to more complex approaches, with coefficients of determination of approximately 1 for individual components as well as on system level. The modelling principle can be transferred to additional components. The presented models can be utilized to perform city district simulations. Thereby, measures like DSM can be evaluated.

In future work, we will extend the media representations to media with temperature dependent properties. Additionally, further components will be added, enabling the analysis of a greater variety of building and district energy systems.

References

- Annex60. IEA EBC Annex 60 Modelica library, 2015. URL <https://github.com/iea-annex60/modelica-annex60>.
- Ruben Baetens, Roel De Coninck, Filip Jorissen, Damien Picard, Lieve Helsen, and Dirk Saelens. OpenIDEAS - an open framework for integrated district energy assessments. In *Proceedings of the 14th IBPSA Conference*, 2015. (submitted).
- Francesco Casella. On the formulation of steady-state initialization problems in object-oriented models of closed thermo-hydraulic systems. pages 215–222. 2012. doi:10.3384/ecp12076215. URL http://www.ep.liu.se/ecp_home/index.en.aspx?issue=76.
- Francesco Casella, Michael Sielemann, and Luca Savoldelli. Steady-state initialization of object-oriented thermo-fluid models by homotopy methods. 2011. URL https://www.modelica.org/events/modelica2011/Proceedings/pages/papers/04_2_ID_131_a_fv.pdf.
- Deutscher Wetterdienst DWD. Aktualisierte und erweiterte Testreferenzjahre (TRY) von Deutschland für mittlere und extreme Witterungsverhältnisse. Technical report, Bundesinstitut für Bau-, Stadt- und Raumforschung, 2011.
- Marcus Fuchs, Ana Constantin, Moritz Lauster, Peter Remmen, Rita Streblov, and Dirk Müller. Structuring the building performance Modelica model library AixLib for open collaborative development. In *Proceedings of the 14th IBPSA Conference*, 2015. (submitted).
- Moritz Lauster, Jens Teichmann, Marcus Fuchs, Rita Streblov, and Dirk Mueller. Low order thermal network models for dynamic simulations of buildings on city district scale. *Building and Environment*, 73:223–231, 2014. ISSN 03601323. doi:10.1016/j.buildenv.2013.12.016.
- Dirk Müller, Antonello Monti, Sebastian Stinner, Tim Schlösser, Thomas Schütz, Peter Matthes, Henryk Wolisz, Christoph Molitor, Hassan Harb, and Rita Streblov. Demand side management for city districts. *Building and Environment (In Press)*, DOI: <http://dx.doi.org/10.1016/j.buildenv.2015.03.026>, 2015. ISSN 0360-1323. doi:<http://dx.doi.org/10.1016/j.buildenv.2015.03.026>. URL <http://www.sciencedirect.com/science/article/pii/S0360132315001432>.
- Christoph Nytsch-Geusen, Jörg Huber, Manuel Ljubijankic, and Jörg Rädler. Modelica buildingsystems - eine modellbibliothek zur simulation komplexer energietechnischer gebäudesysteme. *Bauphysik*, 35(1):21–29, 2013. ISSN 1437-0980. doi:10.1002/bapi.201310045. URL <http://dx.doi.org/10.1002/bapi.201310045>.
- Michael Wetter. *Modelica Library for Building Heating, Ventilation and Air-Conditioning Systems*. Lawrence Berkeley National Laboratory. Environmental Energy Technologies Division and Distributed by the Office of Scientific and Technical Information, U.S. Dept. of Energy, Berkeley and Calif and Oak Ridge and Tenn, 2009.
- Michael Wetter. Modelica buildings library 2.0.0, 2015a. URL <http://simulationresearch.lbl.gov/modelica/FrontPage>.
- Michael Wetter. Buildings library user guide: 2. best practice, 2015b. URL <http://simulationresearch.lbl.gov/modelica/userGuide/bestPractice.html#thermo-fluid-systems>.
- Michael Wetter, Wangda Zuo, Thierry Stephane Nouidui, and Xiufeng Pang. Modelica Buildings library. *Journal of Building Performance Simulation*, 7(4):253–270, 2014. doi:10.1080/19401493.2013.765506.
- Michael Wetter, Christoph van Treeck, and Jan Hensen. IEA EBC Annex 60, 2015. URL <http://www.iea-annex60.org/index.html>.

Open Source Library for the Simulation of Wind Power Plants

Philip Eberhart¹ Tek Shan Chung¹ Anton Haumer² Christian Kral¹
¹TGM Wien XX, College of Engineering, Austria, dr.christian.kral@gmail.com
²OTH Regensburg, Germany, anton.haumer@oth-regensburg.de

Abstract

This paper presents the new open source Modelica library WindPowerPlants. For the economic assessment of either a wind power plant or an entire wind park, the accurate prediction of the energy output is essential. Such prediction is usually performed by means of calculations based on statistical wind data. The proposed WindPowerPlants library is capable of assessing the energy output both for statistical and real wind data based on time domain simulations.

In the presented version of the library wind turbine models are modeled with pitch control. The generator models have variable speed and an optional connector to the mains. The entire library is based on power balance conditions and losses are fully neglected. Yet, the library can be extended towards more detailed models considering different types of losses.

The structure and components of the library are presented. Simulations examples are shown and compared with reference data. The applicability of the proposed WindPowerPlants library is demonstrated and possible enhancements are discussed.

Keywords: Wind power plants, pitch control, variable speed, energy, power, statistical wind data

1 Introduction

Wind power significantly contributes to the total electric energy generation in Europe. Since the economic assessment of future wind power plants is essential, accurate calculations and simulations are needed to predict the energy output of these plants. Therefore, the modeling and simulation of wind power plants is continuously under research. Particular aspects are the wind turbine characteristics (Anderson and Bose (1983); Heier (2009); Ahmed et al. (2014)), the integration of generator models (Mihet-Popa et al. (2004); Yin et al. (2007)), control (Catana et al. (2010); Merabet et al. (2011); Muyeen (2012); Mehdi et al. (2013); Alizadeh and Yazdani (2013)), grid integration (Yuan and Li (2014)) and stability (Du et al. (2014)).

Modelica is particularly suitable to the simulation

of wind power plants due to the multi physical domain approach. In Petersson et al. (2012) vertical axis wind power plants and their control aspects are investigated. A paper on variable speed wind turbine models in power system dynamics simulations is published in Enge-Rosenblatt and Schneider (2008). In Strobel et al. (2011) a Modelica library for offshore wind turbines including structural coupling is presented.

The WindPowerPlants library was developed during a Diploma project at the Technical Engineering College, TGM. For the development of the library OpenModelica was used. The library is published under the Modelica license 1.1 and available through GitHub at <https://github.com/christiankral/windpowerplants>. The main motivations for developing the presented library were:

- Investigate control mechanisms of wind power plants
- Support teaching activities in simulation
- Provide an open source library that may initiate further developments

The proposed library does not model all the controllers that a real wind power plant has. Instead, the intention was to model the overall behavior of wind power plants in such a way, that the major operating conditions are fulfilled. Therefore, only mechanical dynamics are taken into account. Electrical transients are fully neglected. There is yet an electrical interface available to couple electrical networks with one or more wind power plants. This electrical interface to the mains is based on a quasi static multi phase connector.

The WindPowerPlants library is based time domain simulations. Time domain simulations allow a higher flexibility for the investigation of different scenarios than statistical investigations. For example, the impact of time (and temperature) dependent air densities can be investigated. Due to the openness of Modelica, the wind power plant simulation can also be combined with electrical network aspects such as dynamics, stability, etc.

The paper is organized as follows: In Section 2 an overview of the library structure is presented. Section 3 presents the uncontrolled and controlled wind turbine

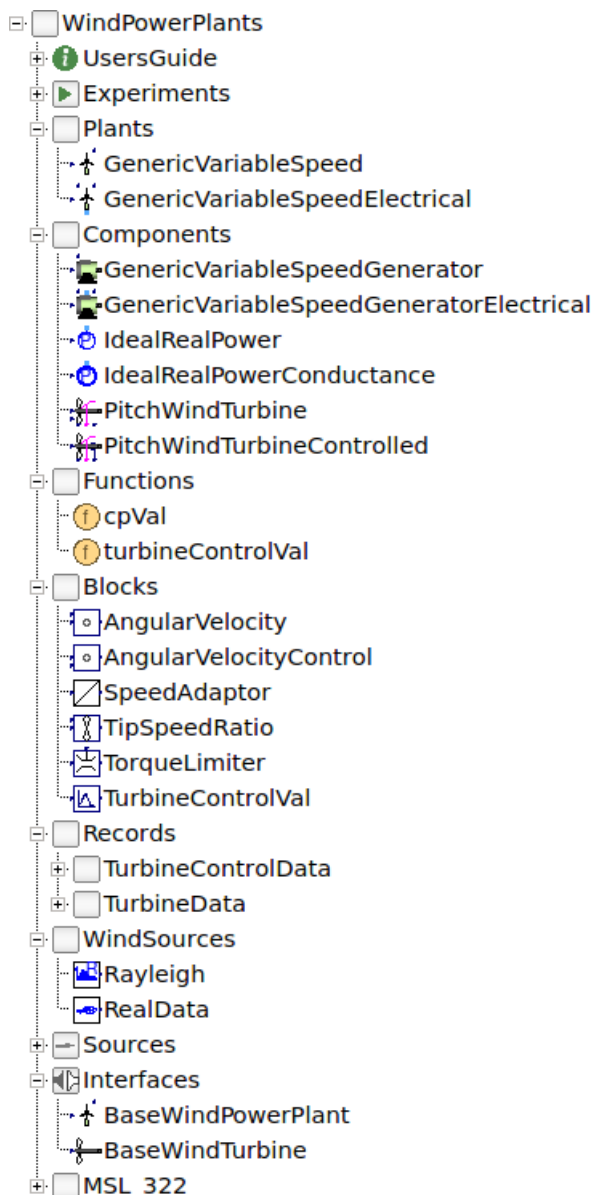


Figure 1. Structure of the WindPowerPlants library

models. The variable speed generator models with and without electrical mains connector are explained in Section 4. In Section 5 the structure of the power plants including the control of the generator is shown. Section 6 presents statistical and real wind data sources and Section 7 compares simulation examples with reference data.

2 Library Structure

The WindPowerPlants library contains models of entire wind power plants including components. Generic plant models with variable speed generators are located the package `Plants`, as shown in Fig. 1.

Package `Components` contains two models of wind turbines, one with pitch angle input, and another with

controlled pitch angle. Until now two generic variable speed generator models are implemented. The first model converts the mechanical power to a power signal connector. The second generator model is equipped with a quasi static three phase connector to the mains and neglects all losses.

In package `Blocks` the controllers, limits, and blocks for the calculation of physical relationships are located. Package `Records` contains data records of different wind turbines and the optimum pitch angles to maximize the power coefficient and thus power output. `WindSources` are the data sources of wind velocity signals.

3 Wind Turbines

For a given wind speed v the total kinetic power of the wind is

$$P_w = \frac{1}{2} \rho \frac{\pi D^2}{4} v^3, \quad (1)$$

where ρ is the density of air and D is the rotor diameter. A wind turbine cannot convert the entire kinetic wind power, but only a fraction. This fraction is represented by the power coefficient c_p . The harvested power is thus determined by the product $c_p P_w$. The theoretical limit of the power coefficient, $c_{p,max} = \frac{16}{27} \approx 0.5926$, was determined by Betz; see Heier (2009). The power coefficient of a real wind turbine is always less than this theoretical limit. The actual power coefficient c_p relies on the design of the wind turbine and, e.g., the pitch angle. In the proposed wind turbine models it is assumed that the pitch angle (or blade angle) β of the blades can be controlled. The pitch angle is the angle of rotation of the blades to control the utilized power. The alternative of stall controlled turbines is not yet included in the WindPowerPlants library. Future wind turbine models may just be modeled by the power coefficient as function of wind speed, since these characteristics are often provided by wind plant manufacturers.

3.1 Turbine Behavior

The power coefficient of wind turbines as function of the pitch angle is usually modeled by empiric equations as provided by Heier (2009):

$$c_p = c_1 \left(\frac{c_2}{\lambda_1} - c_3 \beta - c_4 \right) e^{-\frac{c_5}{\lambda_1}} + c_6 \lambda_1 \quad (2)$$

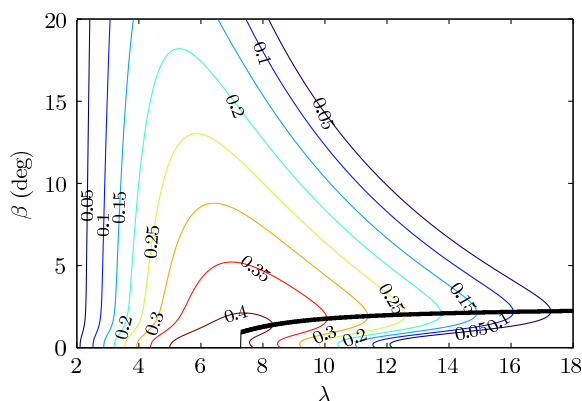
$$\lambda_1 = \frac{1}{\frac{1}{\lambda + 0.089} - \frac{0.035}{\beta^3 + 1}} \quad (3)$$

In these equations c_1, \dots, c_6 are turbine specific per unit coefficients and λ_1 is the internal wind tip ratio which in turn relies on the tip speed ratio

$$\lambda = \frac{\omega D}{2v} \quad (4)$$

Table 1. Wind turbine coefficients of different literature references

	Heier (2009)	Thongam et al. (2009)
c_1	0.5	0.5176
c_2	116.0	116.0
c_3	0.4	0.4
c_4	5.0	5.0
c_5	21.0	21.0
c_6	0.0	0.006795

**Figure 2.** Curves of constant power coefficients, c_p , as a function of the tip speed ratio, λ , and the pitch angle β for the wind turbine coefficients given by Heier (2009)

and the pitch angle β in degrees. The tip speed ratio is the ratio of the peripheral speed over wind speed. The peripheral speed equals the angular velocity ω times half the diameter.

The wind turbine coefficients c_1, \dots, c_6 also reflect the actual geometry of the blades. Examples of turbines from Heier (2009) and Thongam et al. (2009) are compared in Tab. 1. For the wind turbine coefficients given by Heier (2009) curves of constant power coefficients are depicted in Fig. 2. The bold black curve in this figure indicates the curve of maximum c_p for variable tip speed ratio. This curve represents the optimum pitch angle as function of tip speed ratio. The bold black curve is approximated by a third order polynomial function

$$\beta_{\text{opt}} = p_1 \lambda^3 + p_2 \lambda^2 + p_3 \lambda + p_4. \quad (5)$$

The polynomial coefficients p_1, \dots, p_4 are stored in a parameter record matching the turbine model and have to be determined by a pre-processing tool. From Fig. 2 it is also obvious that there is the region for $\lambda < 7.3$ where $\beta = 0$ leads to the optimum power coefficient. For $\lambda > 18.5$ the pitch angle $\beta = 0$ leads to the optimum power coefficient; this region is, however, not depicted in Fig. 2.

The approximation of the power coefficient in (2) and (3) has the drawback that negative pitch angles cause non-meaningful results – even though negative pitch angles are used in practice. However, the actual wind

power shall be limited by positive values to avoid plausibility problems,

$$P = \max(0, c_p P_w). \quad (6)$$

3.2 Uncontrolled Wind Turbine

The signal inputs of the uncontrolled wind turbine model are the wind velocity v and the pitch angle β . The turbine model also has a rotational connector taken from `Modelica.Mechanics.Rotational.Interfaces`. When the wind velocity exceeds a certain threshold, the turbine reaches the power limiting range. In this case the boolean output `limit` becomes `true`. Additional signal outputs are the tip speed ratio, λ , and the angular velocity, ω .

The uncontrolled wind turbine model is coded textually. The most relevant equations are:

```
// Tip speed ratio
lambda * v = w * D / 2;
// Power coefficient of the turbine
cp = WindPowerPlants.Functions.cpVal
    (turbineData, lambda, beta);
// Power of wind
powerWind = 0.5 * Modelica.Constants.pi
    * rho * (D / 2) ^ 2 * v ^ 3;
// Power harvested by power coefficient
power = max(0, cp * powerWind);
// Set boolean indicator if power
// limiting range is reached
limit = power >= powerMax or pre(limit)
    and power >= 0.99 * powerMax;
// Angular velocity
w = der(phi);
// Power balance
power = tau * w;
```

In order to operate the uncontrolled wind turbine model, signal inputs for the wind velocity, v , and the pitch angle, β , are required. In a simulation model of a full wind power plant the pitch angle shall be provided by a controller. Based on the uncontrolled wind turbine model a standard controlled wind turbine model is included in the library.

3.3 Controlled Wind Turbine

The standard controlled wind turbine model provided in the `PowerPlantsLibrary` and shown in Fig. 3 utilizes the uncontrolled turbine model of Subsection 3.2. In this model the pitch angle β is controlled for two different operating ranges of the wind turbine. The two different ranges are indicated by the boolean output `limit` of the uncontrolled wind turbine model:

- optimum power coefficient range: `limit = false`
- power limiting range: `limit = true`

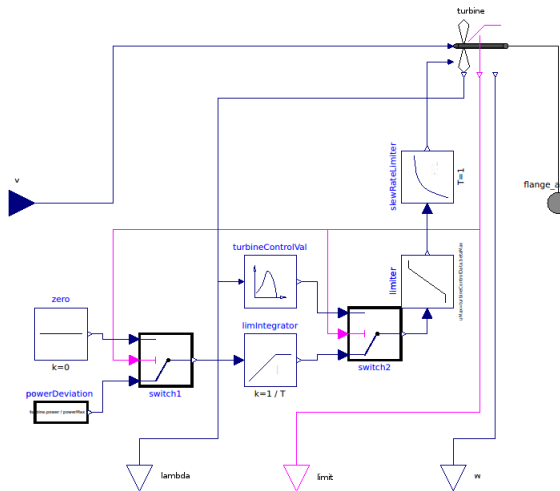


Figure 3. Controlled wind turbine

In the **optimum power coefficient range** the pitch angle β is determined by the polynomial approximation described in Subsection 3.1. This task is accomplished by the block `turbineControlVal` depicted in Fig. 3. The actual pitch angle is then limited by the interval $[0^\circ, 90^\circ]$. After limiting β , a first order delay is used to smoothen the pitch angle response. In a real wind power plant the pitch angle slope is limited. For simplicity reasons, a first order delay is used in the `WindPowerPlants` library.

When the wind turbine is in the **power limiting range**, the pitch angle has to be controlled such way that the actual power P does not exceed the maximum power P_{max} . The maximum power P_{max} is a parameter of the wind turbine model. In the power limiting range the relative power deviation

$$\Delta p = \frac{P}{P_{max}} - 1 \tag{7}$$

is controlled to be zero. For this purpose a fast limiting integral controller is used which processes (7).

4 Variable Speed Generators

The implemented variable speed generators are generic. This means, that typical electrical characteristics of either induction or synchronous generator are not taken into account. The generator models solely rely on power balances and no losses are taken into account.

In the actual implementation the maximum angular velocity of the generator is not limited. However, a real generator does have a maximum angular velocity and frequency, respectively. The difference between the real and the modeled behavior are

- different tip speed ratios,
- different power coefficients,

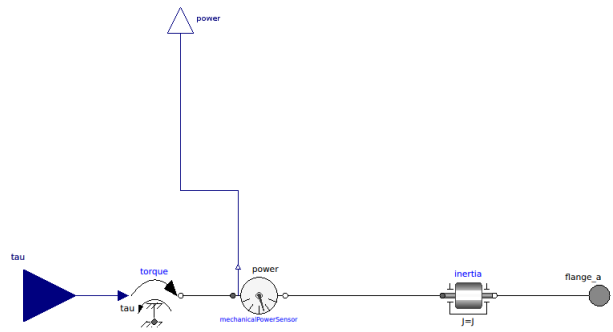


Figure 4. Generic variable speed generator model

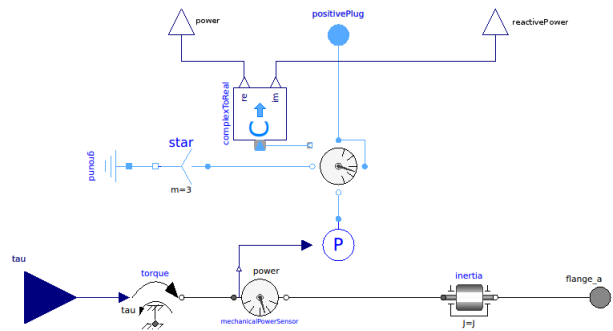


Figure 5. Generic variable speed generator model with electrical connector to the mains

but the output power is the same, since the maximum angular velocity occurs in the power limiting range. Therefore, there is no difference in the total energy output.

Each of the two variable speed generator models shown in Fig. 4 and 5 have one signal input connector: the reference torque. The reference torque is determined by the torque controller of the power plant. In the generic generator models the torque signals are converted into a physical connector quantities. This means for each model, that the reference torque and the actual torque are always equal. A power sensor is used to provide the `power` output connector with the actual mechanical power. The total inertia of the generator is considered.

In the generator model with electrical connector to the mains (Fig. 5) a power balance model is utilized using a quasi static multi phase connector from `Modelica.Electrical.QuasiStationary.MultiPhase.Interfaces`. In the current implementation it is assumed that a grid inverter controls the reactive power Q to be zero. The output of the power sensor is used to operate an ideal electrical multi phase power source which generates real power. In the `WindPowerPlants` library two ideal power sources are provided. The power source `WindPowerPlants.Components.IdealRealPower` used in Fig. 5 utilizes the voltage and current space phasor Haumer et al. (2008). The current space phasor is controlled in such a way that the power input signal of the source and the

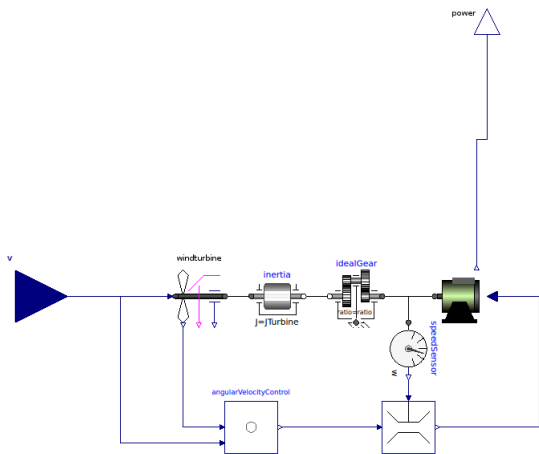


Figure 6. Model of a generic power plant

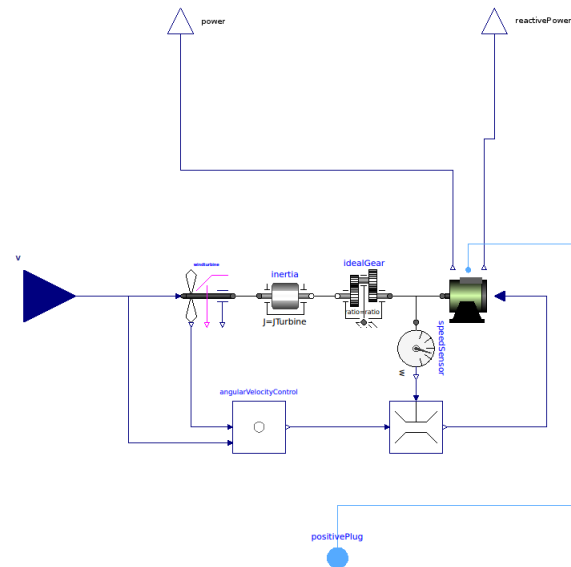


Figure 7. Model of a generic power plant with electrical mains connector

total real power of all phases are equal. This requires the current space phasor to be aligned with voltage space phasor in order to achieve zero reactive power. In the alternative power source `WindPowerPlants.Components.IdealRealPowerImpedance` shown in Fig. 1 the real power is controlled by means of a multi phase conductor to achieve 100 % power balance between the signal input and the three phase electrical power. However, the limitation of $Q = 0$ can be overcome in the future by also controlling the reactive power. By controlling the reactive power the voltage of the wind power plant can be controlled.

5 Wind Power Plants

5.1 Model Structure

The structure of the generic power plant models with and without electrical connector to the mains are depicted in Fig. 6 and 7, respectively. The reason why both models are named generic, is that they are based on a generic variable speed generator.

The input connector of the power plant model is the wind speed. The wind speed is connected with the wind turbine model which converts the fraction c_p of the kinetic wind power P_w into mechanical power P according to (6). The wind turbine model is connected with an inertia, representing the rotating inertias of the wind turbine and the gear with respect to the wind turbine rotational angular velocity. The ideal gear represents the multi stage planetary gear of the real wind power plant. The generator is directly coupled to the high speed side of the gear. The torque input of the generator is controlled by the block `angularVelocityControl`. The torque control is then limited by a rotor speed dependent torque limiter in order to avoid negative rotor speed and high torque dynamics around zero speed.

5.2 Angular Velocity Control

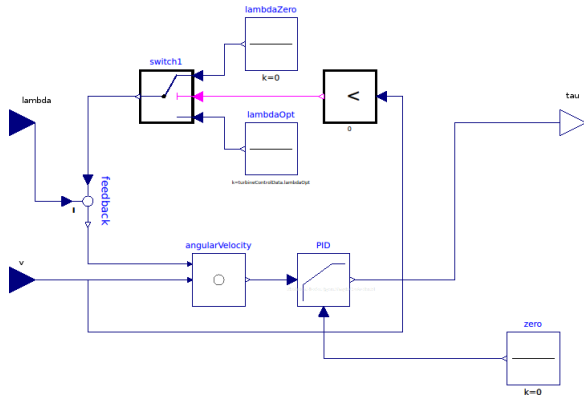
In the angular velocity control of the generator two different **wind speed regions** are distinguished. In the **stand still region**, for wind speeds $v < v_{min}$, the wind turbine is controlled towards zero angular velocity ω_{ref} and thus zero reference tip speed ratio, λ_{ref} . The wind speed v_{min} represents the cut-in speed of the turbine which is in the range of approximately 4 m/s. In the **power generating region**, for $v \geq v_{min}$, regular power conversion occurs. The cut-out speed is currently not considered in the control.

In the implemented control strategy the reference value of the tip speed ratio, λ_{ref} , is set depending on the wind speed region. For the stand still region $\lambda_{ref} = 0$ is chosen. For the power generating region the reference tip speed ratio is derived from the optimum power coefficient as described in Section 3. The optimum power coefficient is the maximum c_p as a function of the actual tip speed ratio; see the bold black curve in Fig. 2. The maximum power coefficient is indicated by two variables, λ_{opt} and β_{opt} . However, as the variable λ_{opt} represents the optimum tip speed ratio, the optimum pitch angle, β_{opt} , is automatically determined from the polynomial approximation of maximum c_p values indicated by (5). This way the control always achieves the highest power coefficient in the optimum power coefficient range.

In Fig. 8 the reference tip speed ratio, λ_{ref} , is the output of block `switch1`. From the reference and actual tip speed ratio the deviation of the reference and actual tip speed ratio

$$\Delta\lambda = \lambda_{ref} - \lambda \tag{8}$$

is calculated. This deviation corresponds with the angu-


Figure 8. Angular velocity control of wind turbine

lar speed deviation

$$\Delta\omega = \frac{2v\Delta\lambda}{D} \quad (9)$$

according to (4). The calculation of (9) is achieved by the block `angularVelocity` in Fig. 8. The angular speed deviation $\Delta\omega$ is fed to the reference input of a PI controller. As the used PI controller has both a reference and a sensor input, the sensor input set to zero, as the difference of these reference and sensor signal are already determined by (9). The output of the PI controller is the reference torque of the generator.

5.3 Torque Limitation

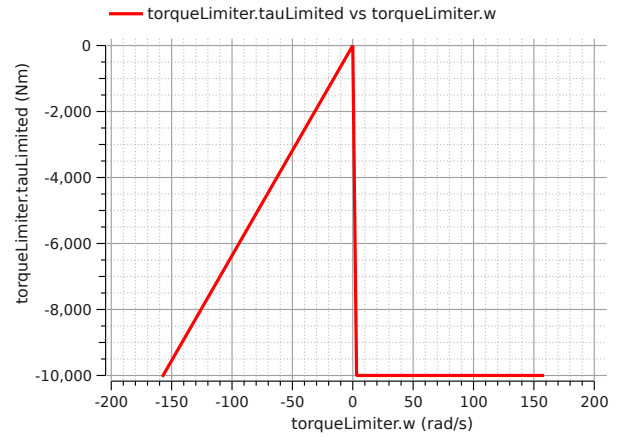
The torque output of the angular velocity control is limited to avoid significant dynamics around zero speed and for negative speeds. Since the electric machine is operated as generator, the mechanical power

$$P_m = \tau\omega \quad (10)$$

is negative. The generator is operated at positive angular speed ω and thus torque τ is negative. Torque limitation, however, is implemented as function of speed. The characteristic of the torque limiter is shown in Fig. 9 for the input torque $\tau = -10\text{kNm}$ and a torque limit of $\tau_{\text{ref}} = 10\text{kNm}$. However, In the investigated case the reference speed $\omega_{\text{ref}} = 50\pi\text{rad s}^{-1}$. For positive speeds greater than 2% of the reference speed the torque is not limited. The range between zero angular speed and 2% of the reference speed torque is limited by the steep linear curve shown in Fig. 9. In the negative speed range – which shall never be reached – torque is limited linearly towards the negative reference angular velocity.

6 Wind Sources

In the `WindPowerPlants` library two different wind sources are provided.


Figure 9. Characteristic of the torque limiter

6.1 Real Wind Data

The first wind data source reads a data file with at least two columns. The first column represents time (in seconds) and the second column is the actual wind velocity (in m/s). The real data wind source model is directly derived by a `CombiTimeTable`, limited to on single parameter: the file name.

6.2 Statistical Wind Data

The second wind source provides the wind speed as function of time, calculated by the discrete Rayleigh distribution

$$d_k = \frac{\pi k \Delta v^2}{2 v_m^2} \exp\left(-\frac{\pi}{4} \cdot \frac{k^2 \Delta v^2}{v_m^2}\right). \quad (11)$$

In this equation k is the interval index, considering n intervals in total. The speed interval

$$\Delta v = \frac{v_{\text{max}}}{n} \quad (12)$$

is derived from the maximum speed and v_m represents the average value of all wind speeds. The variables d_k represent the relative duration of each wind interval with index k . The sum of all distributions is equal to one,

$$\sum_{k=1}^n d_k = 1. \quad (13)$$

For a time domain representation the total time period T is utilized. In a sequence of n different wind speeds

$$v_k = k \cdot \Delta v \quad (14)$$

the duration of the respective wind speed is the determined by $d_k T$. An example of a histogram of Rayleigh distributed wind data is shown in Fig. 10. The respective wind speeds versus time are depicted in Fig. 11. For a particular index k and the accessory wind speed (14) the distribution d_k determines the duration for which the wind speed is constant. After this duration the next wind

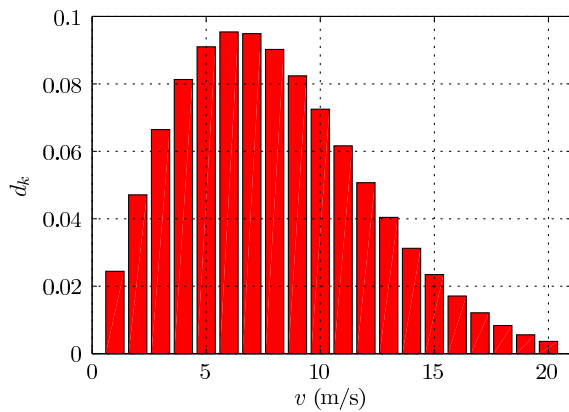


Figure 10. Histogram of Rayleigh distributed wind speed for $v_{max} = 20$ m/s, $v_m = 7$ m/s and $n = 20$

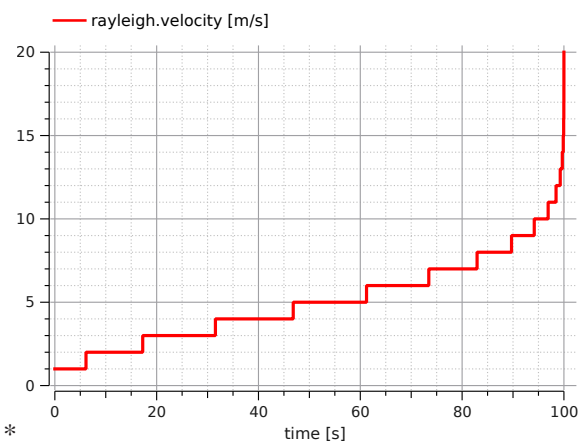


Figure 11. Rayleigh distributed wind speeds in the time domain for $v_{max} = 20$ m/s, $v_m = 7$ m/s, $n = 20$ and a period $T = 100$ s

speed index is processed up to index n . This procedure may be continued periodically. Due to the characteristic of the Rayleigh distribution higher wind speeds and indexes, respectively, give rise for a lower distribution and thus for shorter duration of a constant speed. This is why the curve shows the steepest rise towards the end of the period T .

7 Application Examples

In order to validate the developed wind power plant models, reference calculation data of planned wind power plants are compared with simulation results. An example of a simulation model is depicted in Fig. 6. The reference calculation data used in this paper are provided from the authorities of the province of Lower Austria. Reference data are available from two different wind farms. From the two wind farms altogether three reference power plants are selected. The two wind farms are not explicitly named in this paper to protect wind farm operator data. Instead name synonyms are used. The reference power data were calculated by the appli-

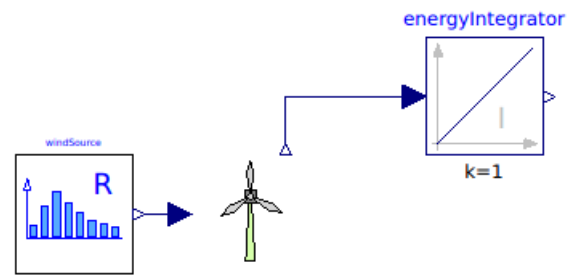


Figure 12. Simulation experiment with statistical wind data

Table 2. General and technical data of the reference wind power plants

		case		
	unit	A	B	C
Wind farm		PD	PD	SP
Plant number		1	2	15
Manufacturer Vestas – type		V112	V112	V90
Power rating	MW	3	3	2
Gear ratio		112.8	112.8	112.8
Rotor diameter	m	112	112	90
Height of hub	m	135	135	125

cants of the wind farms using the software WAsP (Wind Atlas Analysis and Application Program). The general and technical data of the investigated wind power plants and are summarized in Tab. 2. The simulations are performed in OpenModelica on a notebook computer with Intel Core™2 Duo CPU T7250 operated at 2 GHz and 2 GB of main memory.

A comparison of the power plant data, and the reference and simulation data is presented in Tab. 3. The annual energy harvest is indicated as total Energy in MWh divided by one year (1 a), i.e., the physical quantity of power. The deviations of the simulation results are in the range of $\pm 7\%$. The main causes of these deviation are:

- The exact characteristic of the power coefficient of

Table 3. Variables and calculation data of reference (ref) and simulation (sim) wind power plants

		case			
	quantity	unit	A	B	C
	air density ρ	kg/m ³	1.198	1.198	1.198
	average wind speed v_m	m/s	6.95	7.12	7.30
	maximum wind speed v_{max}	m/s	25	25	25
	number of speed intervals n		25	25	25
	interval of investigation	h	8760	8760	8760
	sampling interval	s	10	10	10
	CPU time	s	104	104	104
	power (reference)	MWh/a	10122	10636	6488
	power (simulation)	MWh/a	9497	9905	6869
	absolute power deviation	MWh/a	-625	-731	+381
	relative power deviation	%	-6.2	-6,8	+5.9

the wind turbine is not known; all simulations results are achieved using the wind turbine parameters of Heier (2009)

- The reference data are based on real measured wind speed data whereas the simulations rely on a Rayleigh distribution of the wind speed

Considering on what information is available on the reference power plants, the quantitative simulation results are sufficiently accurate.

8 Conclusion

A new open source Modelica library for the simulation of wind power plants is presented. The library structure and the main components are explained. The control strategies of the pitch angle and the angular velocity of the wind turbine are described. Three reference wind power plants are compared with simulation results. The average power deviations of the simulations from the reference data are in the range of $\pm 7\%$.

References

- S. Ahmed, M.A Rashid, S.B. Yaakob, and A Yusof. Pitch angle control of a grid connected wind turbine. *Intelligent and Advanced Systems (ICIAS), 2014 5th International Conference on*, pages 1–6, June 2014. doi: 10.1109/ICIAS.2014.6869523.
- O. Alizadeh and A. Yazdani. A strategy for real power control in a direct-drive pmsg-based wind energy conversion system. *Power Delivery, IEEE Transactions on*, 28(3):1297–1305, July 2013. ISSN 0885-8977. doi: 10.1109/TPWRD.2013.2258177.
- P.M. Anderson and Anjan Bose. Stability simulation of wind turbine systems. *IEEE Transactions on Power Apparatus and Systems*, PAS-102(12):3791 – 3795, 12 1983. ISSN 0018-9510. doi: 10.1109/TPAS.1983.317873.
- I. Catana, C. A. Safta, and V. Panduru. Power optimization control system of wind turbines by changing the pitch angle. *U.P.B. Sci. Bull., Series D*, 72(1):141–148, 2010.
- Zhaobin Du, Jingshang Chen, Yaopeng Huang, Peng Shen, and Shangyun Liu. The research of simplification of doubly-fed wind turbine in the small signal stability analysis. *Power and Energy Engineering Conference (APPEEC), 2014 IEEE PES Asia-Pacific*, pages 1–6, Dec 2014. doi: 10.1109/APPEEC.2014.7066161.
- Olaf Enge-Rosenblatt and Peter Schneider. Modelica wind turbine models with structural changes related to different operating modes. *Modelica Conference*, 2008.
- 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*, pages 229–236, 2008.
- Siegfried Heier. *Windkraftanlagen: Systemauslegung, Netzintegration und Regelung*. Vieweg + Teubner, 5 edition, 2009.
- Mehdi, Allagui1, Othman B.k, Hasnaoui, Jamel, and Belhadj. Exploitation of pitch control to improve the integration of a direct drive wind turbine to the grid. *J. Electrical Systems*, 9(2):179–190, 2013.
- A Merabet, J. Thongam, and J. Gu. Torque and pitch angle control for variable speed wind turbines in all operating regimes. *Environment and Electrical Engineering (EEEIC), 2011 10th International Conference on*, pages 1–5, May 2011. doi: 10.1109/EEEIC.2011.5874598.
- L. Mihet-Popa, F. Blaabjerg, and I Boldea. Wind turbine generator modeling and simulation where rotational speed is the controlled variable. *Industry Applications, IEEE Transactions on*, 40(1):3–10, Jan 2004. ISSN 0093-9994. doi: 10.1109/TIA.2003.821810. Wind turbine control pitch power torque.
- S.M. Muyeen. *Wind Energy Conversion Systems*. Springer, 2012. ISBN 978-1-4471-2201-2.
- Joel Petersson, Hubertus Tummescheit, Pär Isaksson, and Johan Ylikiiskilä. Modeling and simulation of a vertical wind power plant in dymola/modelica. *Proceedings of the 9th International Modelica Conference*, 2012. doi: 10.3384/ecp12076631.
- M. Strobel, F. Vorpahl, C. Hillmann, X. Gu, A. Zuga, and U. Wihlfahrt. The onwind modelica library for offshore wind turbines – implementation and first results. *Modelica Conference*, 2011.
- J.S. Thongam, P. Bouchard, H. Ezzaidi, and M. Ouhrouche. Wind speed sensorless maximum power point tracking control of variable speed wind energy conversion systems. *Electric Machines and Drives Conference, 2009. IEMDC '09. IEEE International*, pages 1832–1837, May 2009. doi: 10.1109/IEMDC.2009.5075452.
- Ming Yin, Gengyin Li, Ming Zhou, and Chengyong Zhao. Modeling of the wind turbine with a permanent magnet synchronous generator for integration. *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–6, June 2007. ISSN 1932-5517. doi: 10.1109/PES.2007.385982.
- Xibo Yuan and Yongdong Li. Control of variable pitch and variable speed direct-drive wind turbines in weak grid systems with active power balance. *Renewable Power Generation, IET*, 8(2):119–131, March 2014. ISSN 1752-1416. doi: 10.1049/iet-rpg.2012.0212.